Verification Strategy for a RISC-V Core Design

Lliurable de la Fita Final: Memòria final

Autor
Víctor Jiménez Arador

Director
Josué Vladimir Quiroga Esparza (BSC-CNS)

Co-director
Nehir Sonmez (BSC-CNS)

Ponent
Miquel Moretó Planas (DAC)

Grau en Enginyeria Informàtica
Especialitat
Enginyeria de Computadors

20 de Juny de 2019

Resum

Aquest projecte té com a finalitat la definició i l'aplicació d'una estratègia de verificació per poder comprovar el correcte funcionament i adequat respecte a l'especificació d'un nucli d'un processador basat en l'ISA lliure RISC-V. Aquesta estratègia haurà de provar la major quantitat de funcionalitats possibles mitjançant tècniques, software i tests disponibles de forma oberta i alguns programes que s'escriuran expressament per aquest projecte per tal d'automatitzar el procés.

Amb el disseny d'una estratègia de verificació correcta i eficient, s'espera aconseguir trobar i solucionar la major quantitat d'errors possible per tal que abans de la seva fabricació, el processador sigui el més perfecte possible. A més, aquesta estratègia s'aplicarà en un període reduït de temps degut a les necessitats del projecte del processador, pel que ha de ser molt eficaç.

Resumen

Este proyecto tiene como finalidad la definición y la aplicación de una estrategia de verificación para poder comprobar el correcto funcionamiento y adecuado respecto a la especificación de un núcleo de procesador basado en el ISA libre RISC-V. Esta estrategia tendrá que probar la mayor cantidad de funcionalidades posibles mediante técnicas, software y tests disponibles de forma abierta y algunos programas que se escribirán expresamente para este proyecto por tal de automatizar el proceso.

Con el diseño de una estrategia de verificación correcta y eficiente, se espera conseguir encontrar y solucionar la mayor cantidad de errores posible por tal que, antes de su fabricación, el procesador sea lo más perfecto posible. Además, esta estrategia se aplicará en un periodo reducido de tiempo debido a las necesidades del proyecto del procesador, por lo que debe ser muy eficaz.

Abstract

This project has as purpose the definition and appliance of a verification strategy to prove the proper working and accurate to the specification of a RISC-V based processor core.

This strategy must prove the biggest quantity of functionalities possible using techniques, software and tests available and free and some codes that will be written uniquely for this project to automate the project.

With the design of a proper and efficient verification strategy, we expect to find and solve as many errors as possible in order to, before its fabrication, the processor is the most perfect possible. Furthermore, this strategy will be applied in a very short period of time due to the needs of the project of the processor, so it has to be very effective.

Índex

1 Introducció

1.1 Context

Al principi de la informàtica, els microprocessadors es componien de milers de transistors i es dissenyaven a base de col·locar-los de la manera correcta. En l'actualitat però, els processadors tenen milions de transistors, el que en dificulta el disseny.

En aquesta circumstància, van sorgir els llenguatges de disseny RTL (Register Transfer Level)[1] com Verilog[2] i VHDL, amb els quals es defineix el comportament dels diferents mòduls del disseny i les seves interconnexions. Després, utilitzant sintetitzadors del codi RTL, es fabrica el processador dissenyat.
A més, en els últims anys ha sorgit el llenguatge màquina RISC-V[3], que es pot utilitzar lliurement com a base per processadors i que fomenta el desenvolupament de nous microprocessadors.

El procés de sintetització i fabricació del processador és un procés costós i seria un malgast de temps i diners construir un producte que no funciona correctament. Per tant, les empreses d'investigació o que vénen processadors tenen equips molt grans de verificació que s'encarreguen de que el funcionament sigui el correcte.

En els següents apartats es defineixen termes i conceptes relacionats amb l'àmbit del projecte així com els actors implicats en el mateix.

Aquest projecte és un Treball Final de Grau del grau d'Enginyeria Informàtica, de l'especialitat d'Enginyeria de Computadors, de la Facultat d'Informàtica de Barcelona. Ha sigut desenvolupat en col·laboració amb el Barcelona Supercomputing Center (BSC - CNS) [4], com a part del projecte Lagarto, amb l'objectiu d'establir una estratègia de verificació i aplicar-la per tal d'assegurar el correcte funcionament del processador basat en RISC-V actualment en disseny (utilitzant Verilog). Concretament, la estratègia està basada en UVM (Universal Verification Methodology)[5].

UVM és una metodologia basada en System Verilog, un llenguatge que permet verificar el correcte funcionament d'un disseny RTL, i OVM (Open Verification Methodology)[6]. UVM permet instanciar el disseny i un test dins del que s'anomena testbench. Aquest test conté els jocs de proves i uns UVM agents que s'encarreguen d'aplicar els jocs de proves i comprovar el seu correcte funcionament.

En fases inicials de la verificació s'han utilitzat certs aspectes filosòfics de la metodologia UVM. S'han creat jocs de proves i utilitzant simuladors i codis en Python i bash scripts de Linux s'han obtingut i analitzat els resultats d'aquests jocs de proves.

1.2 Actors Implicats

Durant el desenvolupament d'aquest projecte hi haurà diferents persones o entitats que hi tindran impacte o seran afectades per ell. A continuació es detallen els actors implicats.

- **Autor del projecte:** L'autor del projecte, jo mateix, sóc part de l'equip de verificació del projecte Lagarto del BSC. Per tant, el treball constarà de les tasques realitzades en conjunt amb l'equip de verificació i els director i codirector del projecte. Degut al nivell de coneixement dels meus companys i tot els coneixements adquirits durant el desenvolupament del projecte, es podria argumentar que sóc el principal beneficiari del projecte.

- **Director, codirector i ponent:** El ponent és el director del projecte que consisteix en el disseny i producció del processador. Els director i codirector, per altra banda, formen part de l'equip de verificació del disseny i, per tant, el seu paper és semblant al meu. A més, l'ensenyament i guia que em donen durant la realització del nostre treball és indispensable pel correcte desenvolupament del mateix.

- **Altres treballadors del Barcelona Supercomputing Center:** En el desenvolupament de les meves tasques he d'interactuar amb la resta de components del projecte Lagarto, com per exemple els dissenyadors, que també aporten els seus coneixements i ajuda a la realització del projecte.

- **Barcelona Supercomputing Center:** L'usuari o beneficiari final serà el Barcelona Supercomputing Center, ja que una vegada finalitzat el disseny i verificat el seu correcte funcionament, es podrà procedir a la fabricació del chip i la finalització, probablement exitosa, del projecte. A més, si aquest projecte resulta exitós, la infrastructura i els equips creats durant la realització del projecte Lagarto, podran ser utilitzats en futurs projectes, així com el coneixements adquirits pels seus treballadors.

- **Fabricant del xip:** En la realització del projecte Lagarto, l'encarregat d'una de les parts més importants del mateix és el fabricant del xip físic. En el cas d'aquest projecte, l'encarregat de la fabricació en la primera entrega és TSMC, mentre que el de la segona és Global Foundries.

2 Estat de l'art

L'objectiu d'aquest projecte és establir i aplicar una metodologia de verificació pel disseny d'un processador. Tot i que hi ha diferents metodologies ja existents, al BSC no hi ha un equip de verificació format que tingui la formació necessària per utilitzar-les, pel que la idea és adaptar mètodes ja existents a les eines i coneixements dels que es disposen al projecte.

Les metodologies existents, de les quals s'adaptaran alguns aspectes, es detallen a continuació.

2.1 OVM

Com que UVM i OVM tenen un funcionament similar, només s'explicarà UVM que és més complet. Ambdues estan basades en System Verilog, la creació i modificació de tests es basen en els mateixos termes i objectius i tenen una estructura de *testbenches* i funcionament similars.

Una de les poques diferències entre UVM i OVM és que UVM inclou algunes classes o mètodes que OVM no inclou, el que li permet tenir noves funcionalitats que la fa més completa.

Per altra banda, la metodologia UVM es va "inspirar" en OVM, pel que té bastant sentit que s'assemblin tant.

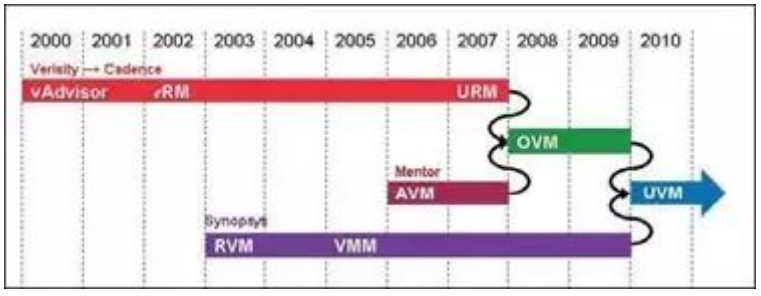

Figura 2: Cronologia de les metodologies de verificació[9]

2.2 UVM

2.1.1 Descripció

Com s'ha explicat abans, UVM és una metodologia de verificació funcional que utilitza System Verilog. Va ser creat per Accellera[7], una empresa de disseny i manufacturació de sistemes automatitzats electronics i circuits integrats.

Es basa principalment en la simulació del comportament del disseny a verificar i tot i que utilitza System Verilog, el disseny pot estar escrit en qualsevol llenguatge descriptiu. A més de simulació, també permet integració d'*assertions*, que comparen els resultats esperats en les diferents senyals amb els esperats en els mateixos.

2.1.2 Funcionament

Per provar el correcte funcionament del disseny, UVM separa els tests a realitzar en diferents tipus. Primerament, s'haurien de fer tests simples i concrets per provar les funcionalitats bàsiques del disseny.
Després, es recomana, per trobar bugs (errors) inesperats, preparar una bateria de tests aleatoris. Això permet crear una gran quantitat de tests que podrien arribar a produir situacions crítiques pel disseny.
Finalment, s'haurien de dissenyar bateries de tests per forçar el disseny al màxim i provar parts específiques que podrien fallar o que siguin crítiques.

A l'hora de realitzar aquestes proves, UVM utilitza tres termes *checkers*, *coverage* i *constraints*. Els *checkers* permeten comprovar que el disseny té un correcte comportament a nivell funcional. Un exemple dels mateixos podrien ser les *assertions*. El *coverage* mesura el nivell de cobertura del disseny, és a dir, el percentatge del disseny que s'assegura que funciona correctament amb els tests realitzats a la verificació. Les *constraints* permeten modificar els tests aleatoris per tal d'assolit el nivell de *coverage* o cobertura necessari. Els tests segueixen essent aleatoris, però estadísticament es poden forçar a "atacar" algunes parts del disseny més concretament.

D'aquests tres termes, el *coverage* és una eina útil per mesurar el nivell d'èxit aconseguit per l'equip de verificació. Per tant, es donen mètodes per calcular-lo i mètriques i ajudes per millorar-lo.

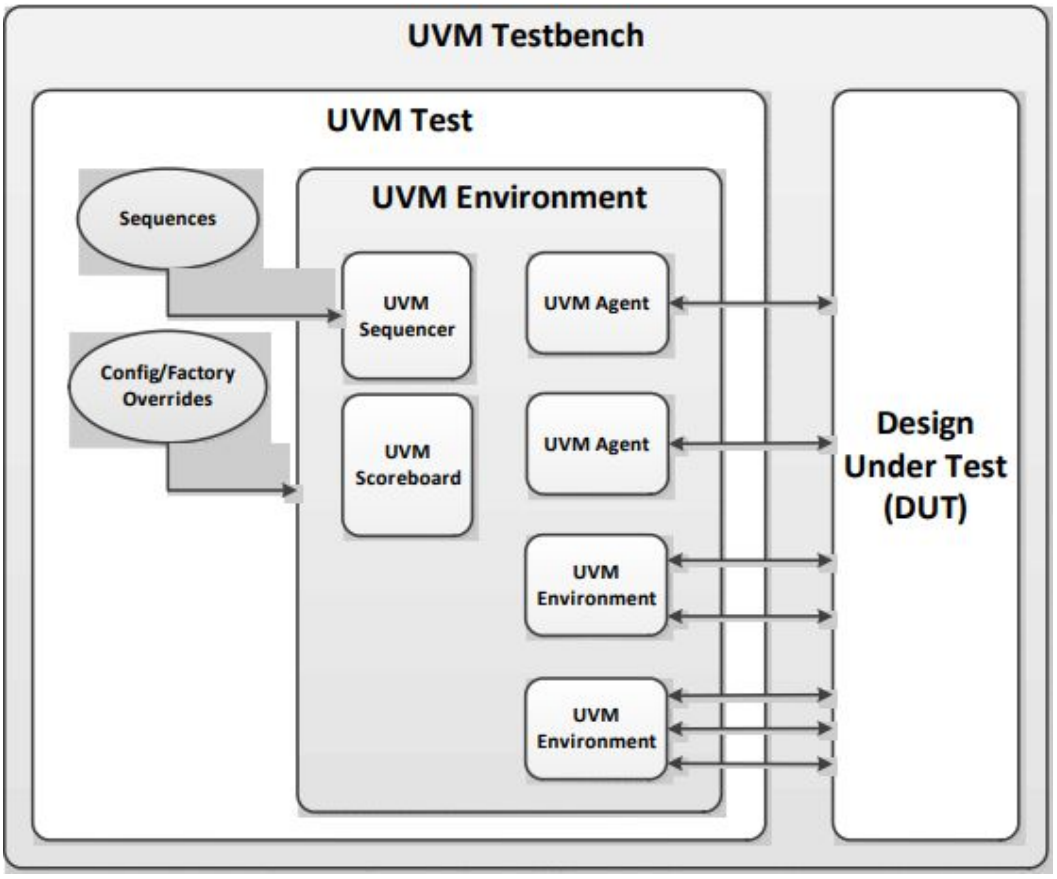

Figura 1: Estructura típica d'un *testbench* UVM[8]

Respecte als altres dos, tenen relació al disseny dels *testbenchs* mencionats a la introducció. Aquests *testbenchs* estan composats pels jocs de proves, uns UVM agents i el propi disseny instanciat.
Els UVM agent, inclourien els *checkers* i introduirien els tests en el disseny mentre que les *constraints* serien incloses als jocs de proves.

2.3 Valoració d'alternatives

Tot i que ambdues alternatives són opcions contrastades i ben documentades, amb exemples exitosos previs i accessibles, no són del tot adequades pel projecte a realitzar.

Per començar, UVM agafa coses de OVM pel que és una metodologia més completa. És interessant conèixer OVM per utilitzar mètodes o aspectes d'aquesta metodologia, però si es disposés de temps suficient i la formació adequada s'escolliria UVM, ja que és més completa. De fet, en projectes futurs és fundamental preparar l'equip de verificació per seguir l'estàndard UVM desde el principi.

Degut als requeriments del projecte, la falta de temps i de preparació per desenvolupar completament i aplicar tots els aspectes de la metodologia, s'ha de trobar una alternativa adequada a UVM. En aquesta alternativa, explicada més endavant, s'adaptaran certs aspectes (principalment) d'UVM al cas del projecte per verificar el més eficientment possible el processador.

3 Abast i Objectius del projecte

En aquesta secció s'especifica detalladament el problema i l'objectiu del projecte i l'abast del mateix, els diferents obstacles que podrien sorgir en el desenvolupament del treball i com sobrepassar-los.

3.1 Formulació del problema

El principal problema que existeix en el projecte Lagarto del Barcelona Supercomputing Center és que tot i que es té dissenyada gran part del processador, encara no s'ha fet una prova extensiva del seu correcte funcionament.
En el moment del començament del projecte, el processador és capaç de fer el *boot* (arrancada) de Linux, però es penja immediatament.

Tot i que es podria anar directament a aquest problema i detectar la causa del mateix en aquesta situació, no és l'aproximació adequada per assegurar el correcte funcionament del processador, ja que podrien sorgir altres errors.

Per tant, s'ha de fer tota la verificació, desde els tests més bàsics fins als més concrets, per finalment arribar a provar el *boot* de Linux. Aquest procés de verificació s'hauria de fer a la vegada que el disseny, podent dissenyar tests específics que els dissenyadors volen provar mentre escriuen el code. La situació actual força a fer aquesta verificació en la meitat de temps que hauria de resultar, pel que encara resulta un repte major.

3.2 Objectius del projecte

Com ja s'ha dit a la introducció, el principal objectiu del projecte és establir i aplicar una metodologia de verificació per comprovar el correcte funcionament, en la major mida possible, d'un processador RISCV.

Tot i això, aquest objectiu inclou d'altres menors que s'han d'assolir per tal d'assegurar que el projecte és un èxit, que es detallen a continuació:

- S'ha de dissenyar i establir una metodologia de verificació funcional que permeti, en el curt període de temps disponible, comprovar el correcte funcionament del disseny donat.

- Aplicar la metodologia dissenyada per comprovar el correcte funcionament del processador.

- Trobar la major quantitat possible d'errors possible i donar feedback detallat als dissenyadors.

- Provar la major part possible del processador buscant errors i regions crítiques del disseny.

- Crear eines per automatitzar processos com l'anàlisi dels tests i dels resultats dels mateixos així com la creació de jocs de proves aleatoris.

3.3 Abast

El projecte, concretament, es centrarà en la descripció de la metodologia de verificació dissenyada i en l'aplicació de la mateixa, no en la resolució dels errors ni en aspectes de disseny. L'abast del projecte, per tant, es redueix a aquests aspectes, essent l'aplicació de la metodologia una tasca en aquest moment impredictible en quant al progrés i al temps que ens prendrà.

Donat el curt període de temps del que es disposa per a la verificació, els resultats esperats no són el nivell òptim de *coverage* (cobertura). Tot i això, es treballarà al màxim per intentar arribar a l'objectiu final, que seria que el processador no tingués cap *bug* o error.

En els aspectes relacionats amb la verificació, no hi ha restriccions respecte a l'abast, ja que el disseny hauria de funcionar perfectament en tots els aspectes. Per altra banda, hi ha seccions del processador més complicades de comprovar que d'altres, pel que

primer s'assegurarà que les funcionalitats bàsiques es comportin correctament. Per això, és possible que alguns errors més complicats no siguin trobats ni solucionats.

A més, una cosa que es tractarà en el següent apartat serà l'aprenentatge de l'autor. Donat que ha d'aprendre a utilitzar algunes de les eines que actualment s'utilitzen al BSC o que s'han d'utilitzar en la nova metodologia, pot ser que el progrés en la verificació es vegi afectat.

Finalment, tot i que es tractarà en l'apartat de metodologia, el procés que es seguirà durant el procés podria fer que alguna part en quedés fora.
Inicialment es faran proves simples per després passar a estressar el disseny, on s'espera trobar errors que s'hauran de localitzar i arreglar per tornar a estressar el disseny per trobar més errors. D'aquesta manera, s'aconsegueix un cicle que requereix un temps per completar-lo, el que podria causar que alguna part del processador no s'avalués completament.

3.4 Possibles obstacles

En el desenvolupament del projecte i la verificació del disseny, poden sorgir diferents obstacles que n'alentin el progrés. A continuació es detallen i es donen solucions pels mateixos.

3.4.1 Falta de coneixement

Donat que és un nou ambient de treball i l'autor no ha utilitzat algunes de les eines que necessita, pot ser que això alenteixi més del compte el correcte desenvolupament del projecte.

Solució: Es posarà el màxim d'esforç possible tant per part de l'autor com dels director i codirector per tal d'adquirir els coneixements necessaris.

3.4.2 Incapacitat per part dels dissenyadors de solucionar els errors trobats

Durant el procés de verificació, es trobaran diferents errors que hauran de ser solucionats per part dels dissenyadors. Aquests errors poden ser evidents o no i pot ser que siguin difícils de localitzar al codi per tal d'arreglar-los. Això pot alentir el procés de verificació ja que una vegada solucionats els problemes, s'hauria de tornar a provar tot el disseny, pel que si es triga en solucionar perdríem bastant de temps.

Solució: Hauríem de ser capaços de localitzar parts del codi que no siguin afectades per a l'anterior bug i treballar en elles paral·lelament per trobar més errors, en cas que hi hagi.

3.4.3 Impossibilitat de replicar els errors trobats

Quan executem grans jocs de prova aleatoris, pot ser que trobem errors al final de l'execució que no siguem capaços de trobar la fracció de test exacta que genera el problema.

Solució: Es dissenyaran codis automatitzats per separar els tests que fallin en tests més petits que permetin trobar la secció del test que falli per tal de donar feedback complet als dissenyadors i facilitar-los el treball.

4 Metodologia

En aquesta secció es detallen tant els mètodes de treball com les eines de seguiment i els mètodes de validació utilitzats en el projecte.

4.1 Mètodes de treball

Com ja s'ha descrit anteriorment, el mètode de treball principal es basarà en la filosofia descrita per la metodologia UVM.
Els mètodes de treball utilitzats, per tant, són:

- A l'inici del projecte, l'autor, ajudat pels director i codirector del treball, aprendrà a utilitzar les eines necessàries i estudiarà el disseny per tal de conèixer-lo a la perfecció. Tot això mentre realitza tasques necessàries pel projecte Lagarto com la redacció de les especificacions del processador. Aquesta metodologia s'anomena ***Learning Orientation***[10], que es basa en l'aprenentatge a mesura que es treballa.

- La metodologia en el procés de verificació serà l'especificada en UVM. D'aquesta manera, s'aconsegueix un cicle en el que s'aplicaran algunes idees de la metodologia ***Agile***[11], com ara les reunions cara a cara amb els companys de grup, l'autosuficiència dels diferents equips i l'autoavaluació i millora del treball realitzat per part dels diferents grups.

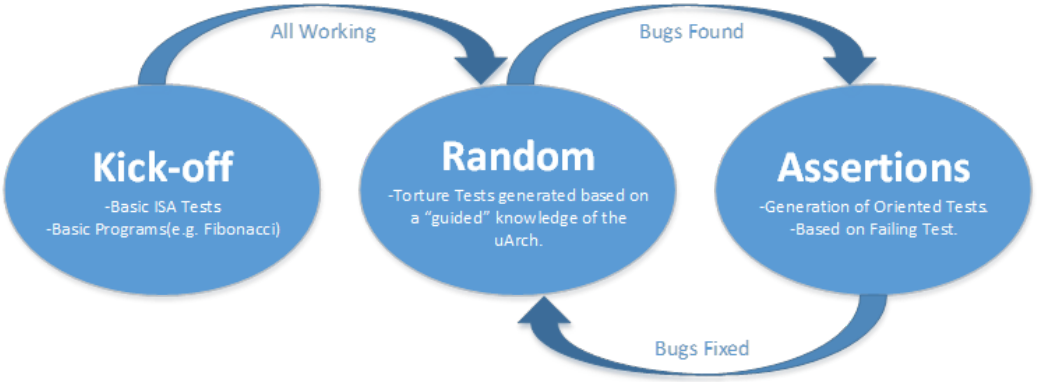

Figura 3: Metodologia del procés de verificació[12]

4.2 Eines de seguiment

Com a eines de seguiments s'utilitzaran, pel codi i els arxius realitzats o produits pel treball realitzat, Github[13] i, en aquest cas, donat que el BSC té la seva propia eina

derivada de Github, Gitlab[14]. Aquesta eina permet tenir versions del mateix codi amb els canvis detallats i taules amb tasques per assignar-les als diferents membres del grups tant de verificació com de disseny.

Per part de l'autor del projecte s'està escrivint un diari on hi ha detalls de les tasques realitzades cada dia per facilitar l'escriptura de la memòria i la documentació així com apunts sobre les mateixes per tal d'explicar-les millor.

4.3 Mètodes de validació

Si el resultat del nostre treball és correcte, cal esperar que el nombre de tests aleatoris que fallen decreixi en cada iteració del cicle. Per tant la millor forma de verificar que la metodologia de verificació funciona correctament és realitzar més tests i comprovar que els errors prèviament trobats estan solucionats.

De la mateixa manera, si el nostre treball no és correcte i l'equip de verificació no funciona de manera adequada, el cicle de treball estarà parat i no donarem feedback a l'equip de disseny, pel que sabrem que la metodologia no és correcta.

5 Bloc inicial de treball

Quan vaig incorporar-me al projecte Lagarto, ja es tenia el disseny del core acabat, s'estava treballant en la periferia del processador i interfícies que utilitzen diferents protocols per afegir més funcionalitats al mateix. Aquestes funcionalitats no són objecte de verificació ja que són interfícies de comunicació que no pertanyen al nucli del processador.

Tot i que ja havia molt treball realitzat, el treball en l'apartat de verificació no era gaire avançat. Com ja s'ha mencionat, la importància de la verificació en el procés de producció d'un processador és molta, pel que era necessari un esforç en aquest aspecte. S'havien realitzat proves bàsiques d'instruccions (sense casos extrems) i s'havia aconseguit fer un boot de Linux, tot i que es penjava en el procés.

Per tant, la situació inicial del projecte és un disseny que té problemes (no funciona correctament) i uns pocs mesos, en els que s'inclou la preparació prèvia necessària, per trobar i solucionar el màxim nombre d'errors possible.

En aquesta circumstància disposem de diferents eines que s'han d'interioritzar per poder utilitzar-les eficientment i que ens permeten fer les primeres passes en la verificació.

5.1 Eines inicials

5.1.1 Verilator

Per poder provar el comportament del nostre disseny, hauríem de construir-lo i donar-li estímuls elèctrics (uns i zeros) a les entrades. Això seria molt costós i contraproduent, ja que s'estaria construint un processador que probablement no funcionaria correctament i que després no serviria per res.

Una altra alternativa seria utilitzar FPGA (Field-programmable Gate Array), un dispositiu capaç de replicar la lògica d'un disseny RTL per fer-ne proves o directament utilitzar-lo.
En el projecte Lagarto s'utilitzen FPGA, ja que permeten simulacions del funcionament real del processador (amb freqüència desitjada), però són dispositius cars i no es disposa de gaires. A més, per produir la "configuració lògica" de la FPGA que emuli el processador desitjat, l'anomenada *bitstream*, es triga bastant i la programació del test no és tan còmode com altres opcions.

Per això es requereix alguna eina capaç de fer el treball de la FPGA de forma més ràpida i barata. Alguns IDEs (Integrated Development Environment) com Xilinx Vivado i

Modelsim permeten fer simulacions dels dissenys però en el cas de la verificació volem tractar el disseny com una caixa negra, pel que ens val Verilator[13].
Aquesta eina ens permet simular el funcionament del disseny i executar tests per comprovar les funcionalitats del mateix. A més, és software lliure com la majoria de les eines utilitzades en aquest projecte, pel que s'adapta a la filosofia del mateix.

El simulador Verilator, entre les moltes funcionalitats que ens aporta, dóna la possibilitat de mostrar a la sortida resultats de les execucions.

5.1.2 Spike

Com ja s'ha mencionat anteriorment, Spike és una eina capaç d'emular el set d'instruccions RISC-V essent capaç de realitzar les mateixes instruccions utilitzant C.
Per tant, no és un simulador d'un processador si no de les funcionalitats del mateix.

A més, permet debuggar (execució pas a pas, contingut dels registres, etc). Aquesta funcionalitat pot ser útil si no entenem el resultat d'una instrucció, però com que disposem de les especificacions de RISC-V, només les hem de seguir.

En el nostre cas, la funcionalitat realment interessant és la que ens permet fer un *dump* (mostrar a la sortida) de la memòria. Això ens dóna l'oportunitat de comparar el resultat de l'execució del nostre processador amb el que seria el correcte.
Addicionalment, mitjançant l'eina de debugging podem fer el dump de la memòria a cada pas, el que ens podria ajudar a trobar l'origen d'alguns errors en combinació amb tècniques que s'explicaran més endavant.

5.1.3 ISA Tests

Donat que RISC-V és lliure, ja hi ha treball realitzat tant en disseny com en verificació.
Una de les eines més utilitzades per verificació bàsica de processadors RISC-V són els ISA tests, que són uns programes en assemblador basats en *assertions* (comprovació de resultats) que proven les diferents instruccions i alguns casos extrems de les mateixes i altres característiques dels processadors com excepcions i memòria virtual.

Aquests tests estan en un repositori a Github i tothom pot tenir accés a ells. És per això que és senzill fer una passada d'aquests tests al inici del procés de verificació per trobar errors bàsics abans de passar a tests més concrets i complexos.

Hi ha tests per 32 i 64 bits i per les diferents addicions d'instruccions com divisions i multiplicacions.

Com ja s'ha mencionat, el funcionament es basa en *assertions*, tenint cada test differents proves d'una instrucció amb diferents operands. Si dins de cada prova el resultat esperat no és el que hi ha *hardcoded*, salta a una rutina d'error i el test es considera fallat.

5.1.4 Torture tests

Tot i que són més importants en següents etapes de la verificació, a la meva arribada ja es disposava de *torture tests*, pel que ja es va treballar amb ells.
Aquests tests, de manera similar als ISA tests es troben a un repositori i són accessibles per a tothom.

Els torture tests, com ja he dit, es tractaran més en profunditat més endavant però mencionar que són tests aleatoris i que consisteixen en una sèrie d'instruccions en assemblador.

5.2 Arquitectura del processador a verificar

Com es diu en apartats anteriors el projecte es realitza en el marc d'un altre projecte del BSC anomenat Lagarto i que té com a objectiu la producció d'un processador que utilitzi l'ISA lliure RISC-V.

En concret, aquest processador té les extensions M (multiplicacions), que afegeix instruccions per realitzar aquestes operacions i és de 64 bits, com a trets característics. Hi ha plans futurs per afegir les instruccions de coma flotant però el disseny a verificar durant el projecte encara no en té.

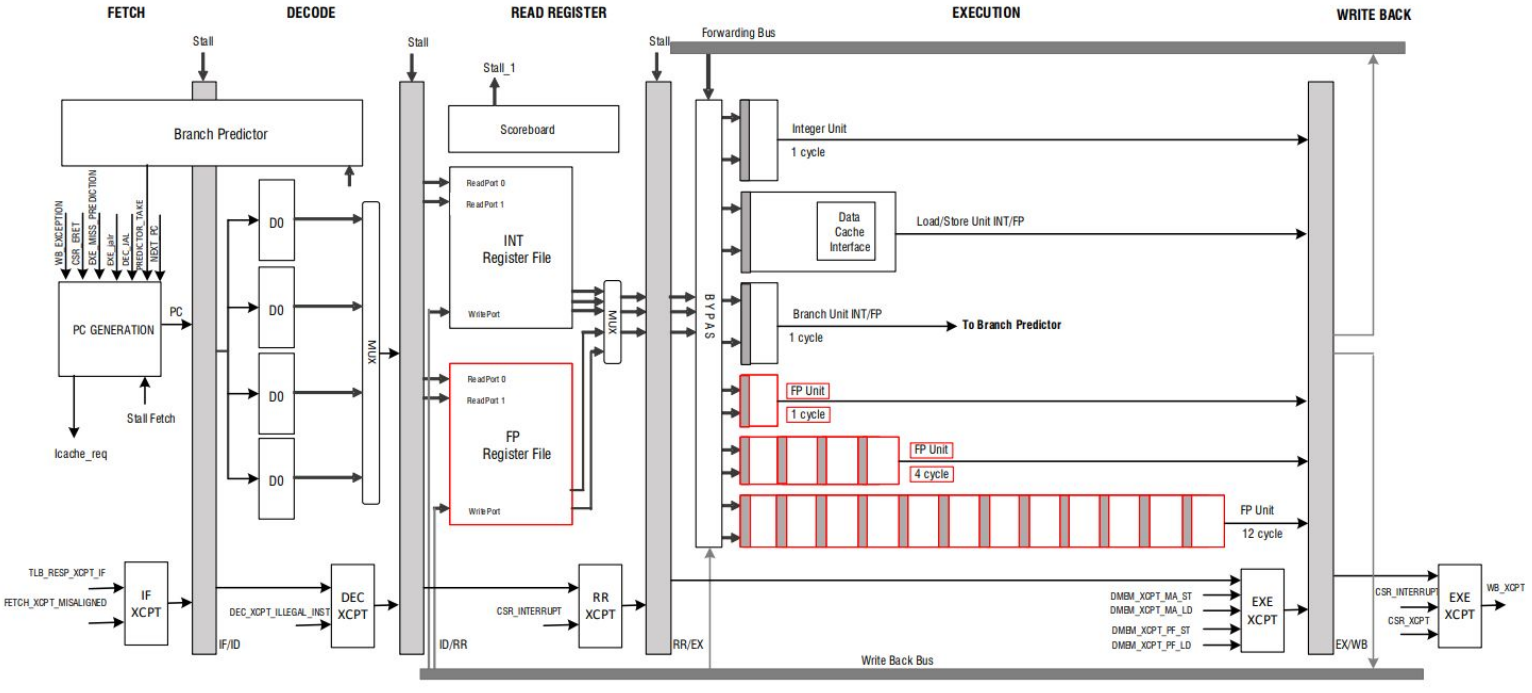

Figura 4: Diagrama de l'estructura de Lagarto[13]

Com a detalls interns rellevants del processador, mencionar que a dins del mòdul principal, el del *core*, hi ha quinze mòduls amb els seus submòduls. A més, és un processador segmentat en cinc etapes; *fetch*, *decode*, *registers*, *execution* i *write back*, com es veu a la imatge. Les etapes *fetch* i *decode* van a memoria d'instruccions per agafar la instrucció indicada pel *program counter* i la decodifiquen, respectivament, mentre que a l'etapa *registers* es recullen les dades per realitzar l'operació pertinyent a l'etapa *execution* i finalment guardar el resultat, si s'escau, a l'etapa *write back*.

Treballa en ordre, tot i que en un futur s'espera afegir la funcionalitat d'*out of order*, que permetrà reduir els bloquejos per dependències de dades al poder executar altres instruccions abans que les bloquejades. Tot i això, disposa d'un predictor de salts que utilitza un comptador saturat i que ajuda a no realitzar tant treball inútil en cas de no realitzar el salt finalment.

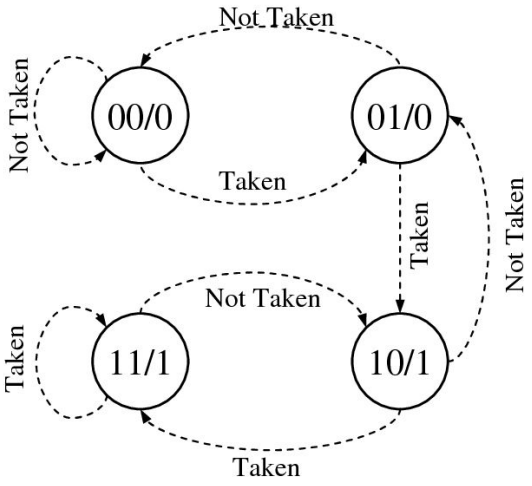

Figura 5: Diagrama representatiu d'un predictor de salt que utilitza un comptador saturat[14]

Per tal que el processador funcioni li calen uns quants perifèrics i per solucionar aquest aspecte existeix un projecte anomenat *LowRISC* que proporciona aquesta estructura. Aquest projecte és la perifèria d'un altre nucli basat en RISC-V anomenat "Rocket" i és un projecte oficial de la RISC-V foundation.

És un projecte obert i que dóna facilitats per integrar qualsevol nucli a aquesta perifèria. És per això que s'aprofita per insertar el nucli de Lagarto i estalviar treball utilitzant la majoria dels mòduls disponibles en aquest projecte.

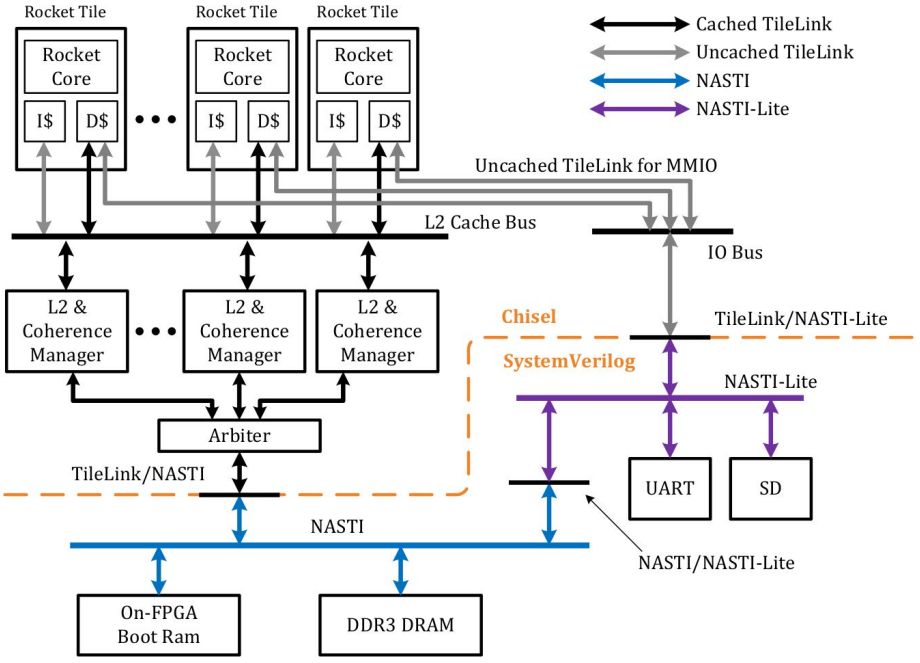

Figura 6: Diagrama de l'estructura de LowRISC[15]

En el cas del projecte Lagarto, s'integra el nucli enlloc de "*Rocket Tile*" i s'adapten les entrades i sortides a les necessàries per poder incluir-lo a LowRISC. Durant la verificació s'està treballant en això i en l'addició d'altres característiques al processador.

Finalment esmentar que tot i que el codi del nucli està acabat i com ja s'ha comentat anteriorment, el processador no és capaç de fer el boot de Linux, però s'espera que durant la verificació es trobi l'error causant d'aquest problema.

5.3 Treball realitzat

Com ja s'ha mencionat anteriorment, al començament del projecte no només feia falta fer la verificació bàsica, sinó que també feia falta familiaritzar-me amb el disseny i conèixer les eines que utilitzaria durant la verificació.

El primer pas va ser durant una setmana realitzar unes petites especificacions del disseny, per poder conèixer-lo a la vegada que treballava amb ell. Per això, vaig haver primer de fer un repàs a Verilog, donat que era la primera vegada que l'utilitzava.
Una vegada que coneixia i entenia el funcionament de cada mòdul del processador, es va passar a la preparació de la verificació.

Es volia poder crear un entorn replicable per poder fer l'execució de tests per part de diversos treballadors i a més permetre el *debugging* dels errors trobats en el mateix entorn per part dels dissenyadors. Per això, es va decidir utilitzar docker, una eina que permet crear un entorn virtualitzat per distribuir i utilitzar aplicacions, per instal·lar les eines de simulació i verificació i permetre l'execució dels tests.

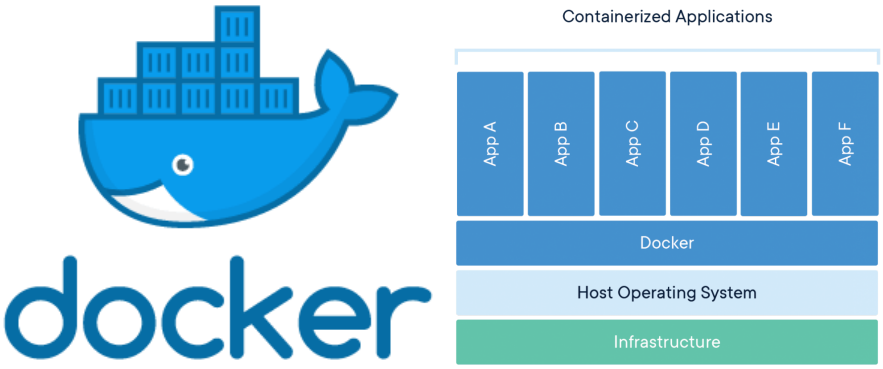

Figura 7: A l'esquerra, el logo de Docker[16], a la dreta un diagrama del funcionament d'aquest software[17], que corre per sobre del sistema operatiu i permet que altres aplicacions corrin dins d'ell, a dins de contenidors.

Tot i que la idea inicialment era crear una imatge amb tot instal·lat i fer la descàrrega directament, finalment es va decidir utilitzar el docker com a entorn però sense descarregar la imatge feta, seguint unes instruccions i utilitzant el repositori comú a Gitlab per mantenir la versió més actualitzada tant del processador com de les eines de verificació.

Per tant, amb l'entorn pensat, el procés de testejat del processador en la primera etapa seria crear la imatge de docker, que inclou llibreries i algun software necessari, crear un contenidor amb la imatge i fer *clone* del repositori del Gitlab.

Un dels submòduls del repositori de Gitlab és el propi codi de Lagarto, en conjunt amb el de LowRISC. D'aquesta manera un dels primers passos a realitzar és compilar certes eines ja mencionades com Spike i Verilator i utilitzar aquesta última per compilar el model del processador i generar l'executable capaç de simular el comportament del processador.

Una vegada es disposa del processador simulat ja es poden fer tests bàsics, com els ISA, que es poden executar directament al simulador i veure si funciona correctament o no. En el cas dels tests ISA disposem de diferents versions dels mateixos i per diferents distribucions de l'ISA. A més estan separats per instruccions i a cada test d'instrucció es proven diferents casos. Això permet fer tests més curts per provar la infrastructura.

Els tests ISA es troben a un repositori amb les instruccions per compilar-los i la seva simulació en Spike (que només serveix per veure que són útils). Bàsicament només calen dues comandes a dins de la carpeta dels test ISA:

- `$ make`
- `$ make run`

Aquestes dues comandes compilen tots els test ISA (inclosos els que no són necessaris, com els d'instruccions vectorials) i posteriorment els executen. Com que no volem executar tests innecesaris, més endavant es crearan llistes de tests per executar.

Segons el propi Makefile dels tests, el resultat dels mateixos es guarda en un arxiu amb el nom del test en qüestió amb l'extiensió ".out" i en la seva documentació també es comenta, en paraules textuals, que són "*silent pass*", és a dir, que només tenen un missatge a la sortida si el test ha resultat erroni. Per tant, si a l'arxiu test.out hi ha algun missatge és que hi ha hagut algun error.

Aquest comportament s'explica mitjançant el mencionat anteriorment; els ISA tests fan *assertions* del resultat en els registres i en cas que siguin diferents als esperats es passa a una rutina que fa un treball extra. Aquest treball extra és crear l'error que es dóna a la sortida de la simulació.

- `$riscv64-unknown-elf-gcc  -DENTROPY=13968 -static -mcmodel=medany -fvisibility=hidden -nostdlib -nostartfiles -I./../env/p -I./macros/scalar -T./../env/p/link.ld rv64ui/add.S -o rv64ui-p-add`
- `$(elf2hex 16 8192 rv64ui-p-add 2> /dev/null || elf2hex 16 16384 rv64ui-p-add) > rv64ui-p-add.hex`

Un exemple de compilació de test seria el següent, en el qual es mostra el resultat de la comanda make anterior pel test de la instrucció "add". Primer es compila el test mitjançant les eines que dóna RISC-V (risv64-unknown-elf-gcc) en un arxiu anomenat com el test sense extensió i posteriorment es tracta aquest arxiu per resultar en l'arxiu test.hex que s'utilitza en la simulació.

- $./`DefaultConfig-sim +max-cycles=100000 +load=rv64ui-p-add.hex`

L'execució del test compilat anteriorment seria com es mostra en la comanda anterior, que seria l'equivalent al "make run" per Spike. En aquest cas DefaultConfig-sim és el simulador compilat per Verilator i l'opció "+load" s'utilitza per especificar el codi a executar.
Per comprovar si el resultat és correcte només s'ha de mirar si la simulació dóna algun missatge d'error.

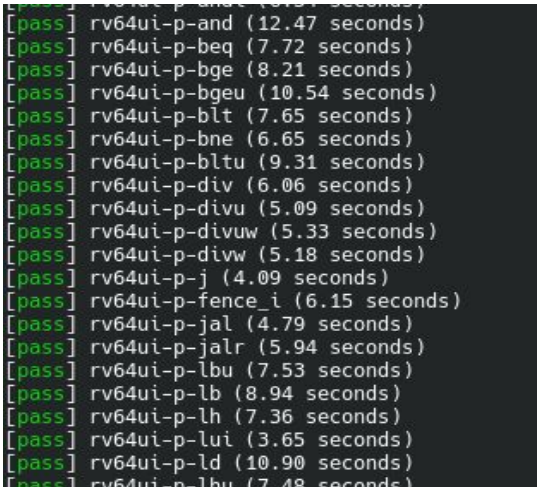

Figura 8: Resultat d'una execució exitosa dels test ISA

Per no haver de fer tot aquest procés cada vegada que es vulgui realitzar un test, alguns companys han realitzat *scripts* que criden aquestes comandes i agafen la sortida de l'execució del test (última comanda) i l'escriuen a un arxiu que posteriorment es llegeix i si és buit el test ha sigut exitós. Per tant, cridant aquest script per un o més tests s'obté directament el missatge *Pass* o *Fail* depenent de si ha anat bé o malament.

En el cas d'aquests tests, Lagarto s'ha adaptat quasi totalment perfecte. Tot els tests van acabar amb resultat positiu excepte els de divisions sense signe.
El problema va resultar ser degut a resultats de divisions que han d'estar *hardcoded*, posats a mà pel programador, en cas que els operands segueixin certs paràmetres com podria ser el cas de les divisions entre zero. Algun d'aquests resultats no estaven posats tal com es diu a les especificacions de RISC-V i no van coincidir amb els esperats al tests.

Condition	Dividend	Divisor	DIVU	REMU	DIV	REM
Division by zero	x	0	$2^{XLEN}-1$	x	-1	x
Overflow (signed only)	-2^{XLEN-1}	-1	–	–	-2^{XLEN-1}	0

Figura 9: Resultats especials de les divisions a l'ISA RISC-V, extrets directament de les especificacions de l'ISA[22].

Tot i això, com que eren errors simples es van corregir immediatament i des de llavors passen tots els tests. Al cap i a la fi, aquest és l'objectiu dels test ISA, trobar errors evidents ràpidament.

Com que el processador segueix sense fer el boot de Linux, sabem que encara té problemes, pel que els test ISA no són suficients per verificar-ne el correcte funcionament. És en aquest punt on es comencen a utilitzar els torture tests, tests aleatoris que són capaços d'estressar el disseny i permeten fer una automatització més completa del procés de verificació.

Tenint aquestes eines es defineix l'estratègia de verificació a utilitzar de cara a la resta del projecte. Aquesta ha de constar d'unes fases que permetin treballar en paral·lel mentre s'arreglen els *bugs* trobats i que s'adapti correctament a les eines de les que es disposen.

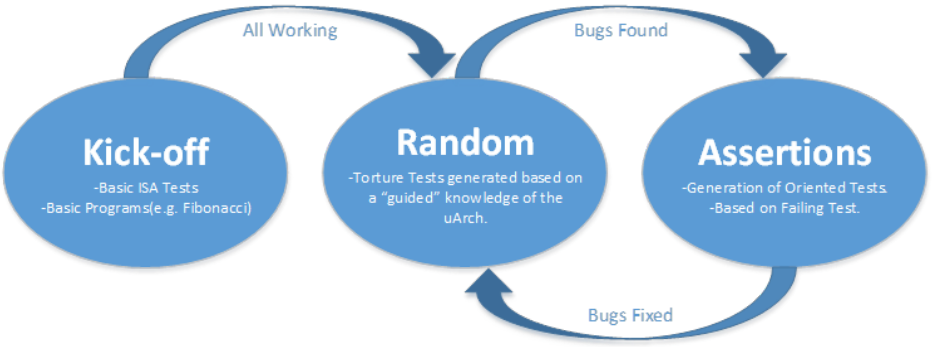

Figura 10: Diagrama representatiu de l'estratègia de verificació dissenyada

Tot i que els torture tests, en aquest punt, semblen més rellevants i eficaços, es considera que és bona idea mantenir els ISA tests per fer les primeres comprovacions sempre que es faci un canvi al processador per tal de trobar algun error causat pels canvis realitzats.
Per tant, es defineix una primera fase en la que s'utilitzen els tests ISA per trobar errors causats per operacions simples per tal de facilitar la localització d'aquests errors.

Una vegada el processador ha passat tots els tests ISA es comença a executar torture tests, que al ser aleatoris poden provar més funcionalitats en més combinacions i amb més instruccions, en definitiva; estressar el processador. Els torture tests ens permetran trobar més errors i automatitzar el procés fàcilment pero no podrem fer proves concretes.

Amb els torture tests, dels quals el procés s'explicarà més endavant, que fallin, tindrem nous errors que s'hauran d'arreglar, pero abans de passar els *bugs* a l'equip de dissenyadors els hem de localitzar i els torture tests no són gaire concrets.
Per aquesta raó necessitem una etapa per analitzar els tests que fallin i escriure tests personalitzats que provin les característiques que donen aquests errors.

Durant el desenvolupament del treball i amb el progrés en la verificació l'estratègia variarà una mica, però inicialment el plan era fer tests bàsics d'instruccions, després una gran quantitat de tests aleatoris per trobar els errors i enviar el feedback més complet possible als dissenyadors per tal d'arreglar-los. Una vegada enviat el feedback, passarem a trobar més errors, a repetir l'etapa de torture tests i una vegada es solucionin els primers errors, es repetirà l'estratègia sencera, assegurant amb els ISA tests que les "reparacions" no han trencat una altra part del processador.

6 Bloc habitual de treball

6.1 Proceso de verificación

Com ja s'ha explicat anteriorment, els torture test són uns codis en assemblador que ens permeten provar les característiques del nostre processador RISC-V. Són una eina bastant útil ja que baixant el repositori i tenint les eines de simulació necessàries pots provar ràpidament i amb diverses configuracions el correcte funcionament del disseny.

El funcionament bàsic, tal i com està explicat en el "README" del repositori, és utilitzar el Makefile per generar-los. Concretament, la comanda a executar és:

- `make igen`

Aquesta opció del Makefile utilitza el codi en el repositori per crear un test en assemblador amb instruccions de RISCV. A més, utilitza un arxiu de configuració amb les següents opcions per donar forma als tests:

```
torture.generator.nseqs      200
torture.generator.memsize    1024
torture.generator.fprnd      0
torture.generator.amo        true
torture.generator.mul        true
torture.generator.divider    true
torture.generator.segment    true
torture.generator.loop        true
torture.generator.loop_size  64

torture.generator.mix.xmem     10
torture.generator.mix.xbranch  20
torture.generator.mix.xalu     50
torture.generator.mix.fgen     10
torture.generator.mix.fpmem    5
torture.generator.mix.fax      3
torture.generator.mix.fdiv     2
torture.generator.mix.vec      0
```

La primera opció permet canviar la mida del test, permetent des d'unes poques instruccions fins a tests bastant llargs, i les "loop" i "loop_size" fan que el test es repeteixi i quantes vegades. Per altra banda "amo" inclou instruccions atòmiques de memòria, "mul" multiplicacions i "divider" instruccions relacionades amb divisions .

Finalment, l'opció "segment" permet canviar l'execució del test. Això resulta en que si es generen tests amb aquesta opció a *true*, els *branch* que hi ha al programa es tracten tots seqüencialment, mentre que si està a *false* els salts es realitzen.

A més, tal i com es pot veure al segon grup d'opcions, es pot escollir la distribució, en percentatge, de tipus d'instruccions al test. En el cas de l'exemple, que és tal i com està la configuració per defecte, hi ha instruccions en coma flotant, cosa no reproduible en el nostre cas.
Per tant, la nostra configuració seria semblant a la de l'exemple però amb les operacions atòmiques de memòria desactivades i totes les instruccions de coma flotant amb un percentatge zero.

Amb la configuració desitjada i utilitzant la comanda "make igen" podem generar un test, però amb un script podríem produir tests el nombre de tests aleatoris que vulguem. Així, tenim una eina bastant automatitzable i personalitzable de generació de tests.

És important conèixer l'estructura i funcionament dels tests generats per entendre el procés de verificació. El modus operandi dels torture test és fer les instruccions necessàries i guardar els resultats als registres per posteriorment fer un bolcat d'aquests a memòria. Aquesta secció de memòria anomenada *signature* contindrà els resultats del test, que podrem comparar amb els correctes obtinguts d'Spike.

Figura 11: Diagrama de l'estructura del codi dels torture tests

El diagrama anterior mostra l'estructura del codi d'un torture test. La part realment important és la del bolcat dels registres a memòria. S'utilitzen *store double* per posar per ordre 31 registres a memòria (els 32 menys el registre x1, utilitzat per guardar l'adreça de memòria necessària a l'*store*).

Aquest bolcat, la *signature*, es pot obtenir tant de l'execució d'Spike com de la simulació mitjançant Verilator. Per tant, la forma de comprovar que el processador ha funcionat correctament seria comparar ambdues *signatures* mitjançant la comanda diff de Linux i veure si són iguals.

Sabent això, podem proposar un exemple d'execució de torture test al nostre entorn de verificació. Per això, suposarem que després d'executar "make igentest" disposem d'un arxiu "test.S" amb el codi del test en assemblador. Primer l'hem de compilar mitjançant les següents comandes:

- `$make riscv_torture test.S`
- `$make riscv_torture test.riscv`

La segona comanda recull el resultat de l'anteriors per produir un nou arxiu. Bàsicament aquestes crides fan el següent treball:

- `$riscv64-unknown-elf-gcc -DENTROPY=28862886 -static -mcmodel=medany -fvisibility=hidden -nostdlib -nostartfiles -Ienv/p -Tenv/p/link.ld test.S -o test.riscv`
- `$(elf2hex 16 8192 test.riscv 2> /dev/null || elf2hex 16 16384 test.riscv) > test.hex`

Com es pot veure, el resultat de la segona comanda és l'arxiu .hex necessari per l'execució simulada del processador. El resultat de la primera (test.riscv) és l'arxiu necessari per l'execució a Spike. Una vegada es disposa dels dos arxius font per l'execució dels tests, es realitzen les següents comandes:

- `spike +signature=test.ref.sig test.riscv`
- `./DirectMappedConfig-sim +max-cycles=1000000 +signature=test.vsim.sig +load=test.hex`

La primera comanda és l'encarregada de l'execució del simulador de l'ISA, del que seria el correcte resultat del test, mentre que la segona és similar a l'execució dels ISA tests, l'execució al processador simulat. En aquest cas però, vam haver de fer un canvi en la configuració de cache de associativa (com en els tests ISA) a mapejat directe, ja que si no no tindriem la *signature* a la memòria al final de l'execució del test.
Ambdues comandes accepten l'opció "+signature" per volcar la memòria (no tota, només la secció declarada al codi) en un arxiu, test.ref.sig (reference) en el cas d'Spike i test.vsim.sig (Verilator simulator) en el cas del processador simulat.

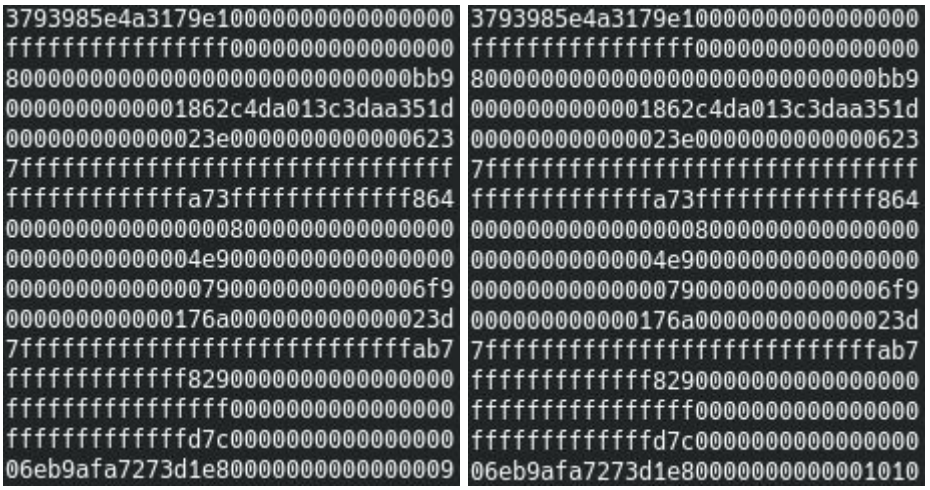

Figura 12: A l'esquerra, la *signature* resultant d'Spike, a la dreta la resultant de Verilator.

Les anteriors serien les *signatures* resultant d'un test erroni. Cal recalcar que el processador consta de 32 registres i la part de la *signature* que conté els registres són 16 línies, pel que hi ha dos registres per línia, estant el registre amb número més elevat (imparell) a l'esquerra i el més petit a la dreta (parell). Les línies seguirien aquest patró; línia 1: x1, x0, línia 2: x3, x2, etc.
Tenint en compte això, podem saber que en el cas d'aquest exemple a la última línea (registres x31 i x30) hi ha un error al registre x30 ja que a Spike el resultat és 0x0000000000000009 i al nostre disseny ha sigut 0x0000000000001010, això ens permetrà buscar al codi on es produeix l'error i analitzar en profunditat la instrucció que la produeix.

Fer aquest procés a mà és una mica tediós, pel que utilitzant la comanda "diff" de Linux podem saber si hi ha alguna diferència. La comanda diff de Linux no dóna cap sortida si els dos arxius són iguals, pel que si hi ha algun *output* sabem que el test ha fallat. Això ens permet automatitzar l'execució de molts tests i donar resultats ràpids. La comanda a realitzar en el cas d'aquest exemple és la següent:

- `diff test.ref.sig test.vsim.sig`

En el cas del test erroni anterior la sortida seria la de la imatge següent:

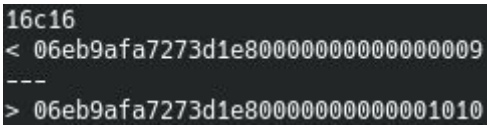

Figura 13: Sortida de la comanda diff de Linux per les *signatures* anteriors

La sortida mostra que en la línia 16 (diff va de 1 a 16, no de 0 a 15) de la *signature* (registres x31 i x30) hi ha una diferència. Aquesta característica ens permet crear un codi que ens busqui directament on està l'error per tal de fer l'anàlisi amb més facilitat.

El codi en qüestió, escrit en Python, simplement agafa l'arxiu ".diff" de l'execució i en llegeix els primers caràcters, fins que troba la primera "c", per localitzar la primera línia on hi ha un error.

Una vegada es té la línia, s'analitzen les línies, correcta i incorrecta, per tal de veure si l'error està a l'esquerra o a la dreta del setzè caràcter amb l'objectiu de saber quin dels dos registres de la línia és el de l'error.
Després es calcula quin registre és mitjançant l'operació: (número de línia - 1) * 2 (dos registres per línia) i es suma 1 si l'error és a l'esquerra del setzè caràcter.

Així, en el cas anterior el càlcul seria: línia 16 menys 1, 15 per 2; 30 i com que l'error és a la dreta no es suma res. Per tant l'error és al registre x30.
Sabent quin registre provoca l'error és dóna un breu missatge que conté el número del registre i es busca en el codi l'última aparició d'aquest registre com a registre destí d'una instrucció i es dóna a la sortida.

Això permet en una execució de test i d'aquest programa saber si falla, quin registre falla i a ón es modifica per última vegada.

Mitjançant l'explicat fins ara, tenim les eines per executar els torture tests, saber si ha resultat erroni o correcte i, si escau, trobar el registre i quina instrucció l'ha produït. Una vegada es disposa d'això, la idea és fer l'execució de grans quantitats d'aquests tests aleatoris. Per això es creen codis, *scripts*, capaços de generar els tests que es vulguin i fer-ne la compilació, execució i comparació dels mateixos.

Durant les primeres setmanes d'ús treballem amb 1000 tests i en surten al voltant de 600 correctes. No és un resultat gaire dolent però no és fàcil d'interpretar degut a que els tests són seqüencials i amb moltíssimes instruccions, no se sap ben bé el que passa.
És en aquest punt quan s'analitza més en profunditat el funcionament dels torture tests i es determina que la complexitat dels mateixos és realment bastant baixa, ja que com ja s'ha mencionat són seqüencials i molt llargs, però les instruccions del principi del test no tenen repercussió en les del final.

És a dir, els resultats de les operacions poden ser sobreescrits en les instruccions següents. Això vol dir que els errors no són arrossegats durant tot el tests sino que són de les últimes instruccions (es carreguen immediats i es fan *loads* a memòria per agafar noves dades) i a més que pot haver més errors durant el tests que no apareguin al resultat final perquè són sobreescrits. Per tant, només tenim les últimes modificacions com a font del testejat i a més perdem eficiència al realitzar un test d'unes 400 instruccions per només evaluar el resultat d'unes 50 al final.

Tot i aquesta "pèrdua de crèdit" dels torture test com a proves fiables, sabem que una gran part d'ells falla, i ens interessa arreglar aquests errors.
Per assegurar l'efectivitat dels torture test s'utilitzen dos solucions, la primera senzilla i sense treball extra i la segona mitjançant un codi i la modificació dels tests.

6.1.1 Tests curts

La primera opció és canviar la mida dels tests utilitzant les configuracions que ens ofereix el generador de torture tests. Canviant l'opció "nseqs" per un nombre més baix, es pot aconseguir un test de molt poques instruccions. És cert que un dels punts a favor dels torture tests era estressar el processador amb moltes instruccions, però ja que fallaven utilitzant només les últimes dels mateixos, la hipòtesi era que es podrien reproduir els mateixos errors si féssim tests de poques instruccions.

Aquests tests més petits tenen més beneficis; donat que tenen poques instruccions, no només són més fàcils de "debuggar" per trobar l'error sinó que a més es generen, compilen i executen de forma molt més ràpida, el que facilita l'execució massiva dels mateixos.
Tenir més tests provoca que estadísticament puguin aparèixer més errors degut a la generació aleatoria.

Com que el test no es composa exclusivament de les instruccions a testejar, si el test té 4 instruccions respecte a les 400 que podria tindre prèviament, l'*speedup* (la millora en temps, temps antic entre temps actual) no és 100. Tot i així si abans es trigava de mitjana uns 25 segons per test en generar i executar, amb els tests petits es triga al voltant de 5 segons.
Amb aquesta millora en temps, es generen diverses tandes de 1000 tests de forma molt més ràpida i en aquest cas fallen de l'ordre de 40 a 50 per tanda.

LLavors, es disposen d'uns 45 tests de mitjana en els que en un interval de com a màxim 10 instruccions hi ha un error. Aquests números són més assequibles per fer-ne l'anàlisi.

Inicialment, es vol fer un anàlisi de la proporció d'instruccions i veure si cal canviar les opcions per fer tests més complets. Es crea llavors un codi capaç de comptar les instruccions dels tests.

Inicialment el codi és molt simple, ja que simplement travessa el codi del test en assemblador i en llegeix el text de les instruccions. Al programa se li passa com a argument un test en assemblador i en crea un arxiu ".txt" al que consten només les estadístiques de les instruccions (nom, número i percentatge) que hi apareixen.

Es disposa d'una llista d'instruccions i per cada una de la llista es fa un *count* de Python i s'en treuen el número d'aparicions. Això no és perfecte ja que hi ha instruccions com "add" i "or" que quan es fa el *count* també es tenen en compte les aparicions "addi" i "xor", el que distorsiona les estadístiques reals dels tests. Aquestes instruccions ja són les que més apareixen pel que no crea un gran problema, però més endavant es millora el programa.

La millora en qüestió passa d'utilitzar el test en assemblador a directament les instruccions.
Per això, és fa un *object dump* (s'obté el codi directament instrucció per instrucció amb aquestes en hexadecimal) mitjançant les eines que ofereix RISC-V, i s'escriu un arxiu amb aquest.

Es crea una llista d'instruccions que conformen una màscara i es van llegint i fent un *count* al test. Es passen les instruccions a binari i utilitzant caràcters *wildcard* es troben les aparicions de les instruccions.
D'aquesta manera, com que el codi d'operació és diferent per totes les instruccions, es solucionen els problemes de l'altra versió.

De totes maneres i fins que aquesta millora no s'implementa, tot i que no sigui perfecta, s'utilitza la versió basada en text. Tenint la deficiència anterior comentada en compte, el programa serveix pels propòsits pels que s'escriu i és la versió que s'utilitza durant la major part de la verificació.

Els resultats del codi comentat anteriorment no són gaire esclaridors, ja que les proporcions són les esperades. En aquest punt, i tenint la possibilitat de calcular el nombre d'instruccions, sorgeix la idea de buscar tendències entre els tests que fallen i els que no. La hipòtesi és que el que pot passar és que en els més o menys 45 tests que fallen hi hagi sempre la mateixa instrucció i per aquesta falli.

Per això es crea un *script* capaç de separar en dos carpetes els tests que funcionen i els que fallen i posteriorment aplica un nou programa que compta les instruccions de tots els tests de la carpeta.

El programa utilitza el codi que compta instruccions basat en text, però en aquest cas és capaç de fer un comptatge de les instruccions totals d'una carpeta i crear un arxiu amb els resultats totals.

Amb una carpeta pels tests que fallen i una altra pels que no fallen, si s'aplica el programa de comptatge total de les instruccions, per cada un dels grups es té la llista d'instruccions que hi apareixen.

Per tant, es crea un altre programa que llegeix les instruccions dels tests que passen i les dels tests que fallen i cada instrucció de la segona llista que apareix a la primera queda eliminada.
Al finalitzar aquest procés, a la llista només queden les instruccions que són exclusivament als tests que donen errors i que mai són a un que no falli. Per tant, és bastant probable que siguin les causants dels errors.

Utilitzant aquest mètode vaig aconseguir veure que a tots els tests que fallaven hi havia sempre una instrucció relacionada amb divisions, ja sigui divisió, divisió sense signe, residu, residu sense signe i les seves versions per 32 bits (divw, divuw, remw, remuw).

Aquesta tendència va evidenciar que hi havia algun problema en el mòdul encarregat de les operacions de divisió.

Encara que hi hagi aquesta estadística, no es pot asegurar que els errors els produeixin aquestes instruccions, pel que es passa a mirar una mostra dels tests erronis i comprovar que els registres amb error són els modificats per aquestes instruccions.
Efectivament, es va confirmar que els errors eren produïts per aquestes instruccions i es va enviar el feedback als dissenyadors.

Més endavant els dissenyadors van trobar la raó del problema. Aparentment, quan les divisions tenien el dividend més petit que el divisor i quan un dels dos era negatiu, les divisions eren errònies.
Els operands es podrien haver comprovat abans d'enviar el feedback per facilitar el treball als dissenyadors, però no es va fer. Aquest és un punt a millorar tant en futures verificacions del projecte i en altres projectes.

6.1.2 Separació dels tests

Paral·lelament a l'execució dels primers tests amb la configuració de poques instruccions, es va treballar en la segona opció per facilitat el *debugging*. En aquest cas, la idea era, ja que els tests eren llargs i es perdien resultats pel camí, fer diverses "divisions" del test.

Inicialment, l'enfocament del programa era mitjançant la creació de X (essent X el nombre de parts en les que es volia dividir el test) subtests amb les diferents parts del codi del test i una *signature* per subtest.
Es va veure, però, que es creaven molts arxius per cada test que es volia analitzar i quasi es multiplicava el temps d'execució pel nombre de subtests degut al *overhead* que provoquen les parts inicials i finals del tests. Per tant, es va canviar l'enfocament del codi.

El nou codi es basa en el bolcat dels registres no només al final del programa sinó també en més punts per crear més checkpoints.

Per això el programa agafa la secció del codi que fa el bolcat dels registres i la replica tantes vegades com se li passi com a argument.
Per que aquest bolcat es vegi reflexat a la *signature* també s'ha de declarar a la part de dades junt a la declaració ja existent, pel que es replica tantes vegades com *checkpoints* es vulguin.

El funcionament del codi és bastant simple, ja que simplement copia el text de les seccions, en canvia el nom en la declaració de les seccions de dades i en canvia l'adreça dels *stores*.

Utilitzant el codi descrit s'obtenia una estructura dels tests semblant a la següent:

Figura 14: Exemple d'estructura d'un test separat en 3 subtests

Mitjançant aquesta modificació es va confirmar que es sobreescrivien els registres i es van trobar errors extra als tests que ja fallaven i la gran majoria dels que abans no tenien errors van passar a ser erronis.
Això es devia a que pràcticament tots els tests contenien errors (encara no havíem trobat la causa dels errors), però no sempre quedaven reflexats en la *signature* degut a la sobreescriptura.

El resultat del test exemple anterior, en quant a la *signature*, és similar al diagrama següent:

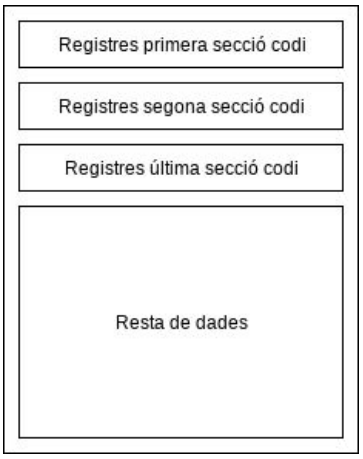

Figura 15: Exemple d'estructura de la *signature* d'un test separat en 3 subtests

Cada secció de registres la comprenen 16 línies amb dos registres cada una, pel que per exemple el registre x4 de la segona secció de codi seria a la línia 19, a la meitat dreta. Per facilitar el procés de *debugging* es crea un programa senzill que troba els errors.

El programa mencionat simplement llegeix el ".diff", de manera similar al programa que buscava el registre, però en aquesta ocasió mitjançant la línia de l'error també es determina en quina partició del codi es troba l'error.

Com que cada bolcat de registres suposa 16 línies, es divideix la línia de l'error entre 16 i s'obté en quina secció està (entre 0 i X-1, essent X el nombre de particions).
Per exemple, si la línia de l'error és la 36 i l'error és als 2 bytes amb menys pes a la línia, l'error és a l'*split* 36 entre 16; 2 (el tercer), a la quarta línia d'aquesta secció i al registre (4 - 1) * 2 = 6 + 0 (perquè és a la dreta); 6.

Per tant, utilitza un mètode semblant al del codi explicat anteriorment i per l'exemple anterior per trobar l'error s'hauria d'anar a trobar l'error a la tercera partició de codi a la última modificació del registre x6 abans del bolcat.
També es dóna a la sortida aquesta instrucció, de la mateixa manera que amb el programa base.

La idea darrera d'aquest codi és trobar errors intermitjos en tests que no fallin sense aplicar-li la modificació de les particions, però també funciona per tests que fallin inicialment. El problema és que en aquests casos només dóna el primer error, però qualsevol error trobat ens és rellevant, per suposat.

Durant la fase d'anàlisi dels tests dividits, es va trobar el problema amb les divisions mitjançant els tests curts. Es va comprovar manualment i es va confirmar que els errors trobats en els tests llargs mitjançant la modificació descrita eren causats per les divisions.
En setmanes posteriors es va corregir l'error de les divisions i tots els torture tests van funcionar sense cap error.

Aquest va ser el gran problema d'aquesta opció: que l'altra va ser més efectiva. Personalment penso que l'ús de tests més llargs permet estressar i trobar més errors, pel que és millor. Estadísticament és cert, ja que generant 100 tests petits hi ha una fracció molt petita de les instruccions que hi haurien generant 100 tests normals.
A la pràctica però, va ser més eficient l'execució de multitud de tests curts ja que vam trobar i arreglar els errors que podíem trobar mitjançant els torture tests simplement amb aquests.

Si al disseny hi haguessin problemes que no tinguessin relació amb les operacions sinó amb el pipeline, els tests llargs i aquests *checkpoints* ens permetrien trobar el punt del test on tot falla.

Amb tot, els torture tests han estat molt útils ja que han servit al seu propòsit. Des de l'execució inicial dels tests ISA i haver trobat els errors en les divisions, el progrés dels torture test ha fet possible arreglar d'altres errors que fan que ja no hi hagi cap error en els tests.

Quan es van començar els torture tests els podíem generar, compilar i executar. Quan vam començar a tractar amb les *signatures* vam poder comprovar si el processador passava els tests o no mitjançant la comparació.
Aquest treball ja s'havia fet anteriorment i de fet és per com estan pensats els torture tests, però es va generar el codi de localització de registres erronis, que automatitza més el procés i és la primera aportació exclusiva del projecte.

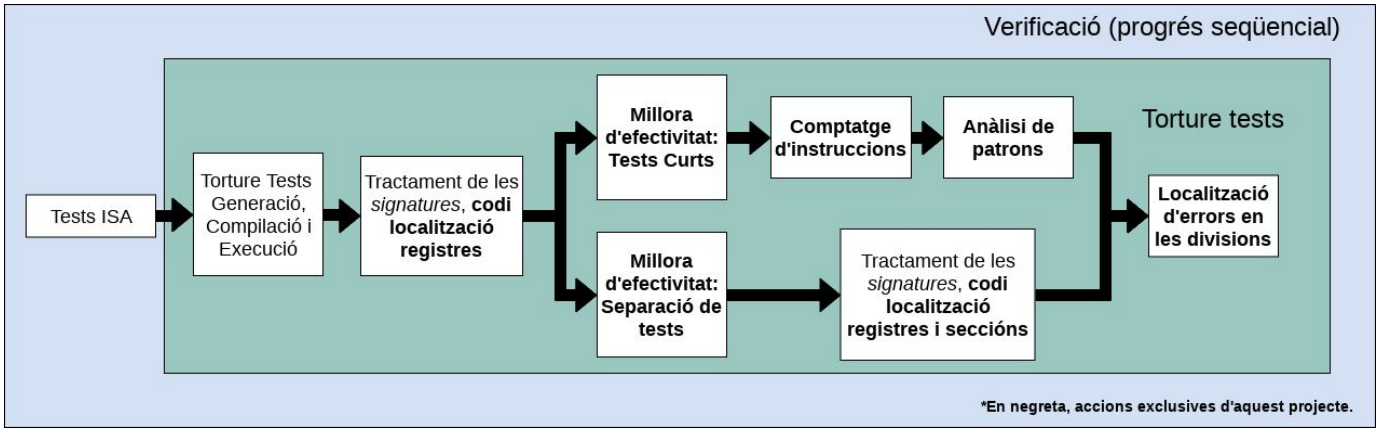

Figura 16: Progrés de la verificació fins la localització dels errors en les divisions

Posteriorment, per millorar l'efectivitat dels torture tests es van pensar les dues solucions ja comentades. La primera, els tests curts, permetia fer el *debugging* més fàcilment i conjuntament amb el codi comptador d'instruccions i l'anàlisi de patrons facilitat per les dades obtingudes van permetre veure la tendència de les divisions fallides.
La segona, la separació de tests en subtests, i la localització dels errors i quines instruccions les generaven a les diferents seccions va confirmar aquesta tendència.

Tot i que ja disposàvem de certes eines, com els tests en sí i els simuladors/emuladors, l'aportació en quant a automatització i facilitació de la verificació

d'aquest projecte ha estat molt gran, com es veu en els programes creats i gràcies als quals, després de trobar i arreglar les divisions, ja no hi ha errors a cap test.

7 Resultats

7.1 Resultats de la verificació

Una vegada explotat tot el potencial dels torture tests, generant milers i milers de tests que durant les últimes etapes del projecte ja no trobaven cap error, el procés de verificació factible del processador ha finalitzat.
Com ja s'ha mencionat, s'han trobat una quantitat rellevant d'errors, tot i que estan molt concentrats en unes poques funcionalitats.

Durant la primera etapa de la verificació, la designada com a tests bàsics, i en la primera passada de l'estratègia, utilitzant els ISA tests es van trobar problemes en els encarregats de provar les divisions. Com que aquests tests diuen quina instrucció ha fallat, vam poder veure ràpidament que els errors eren relacionats amb resultats especials de les divisions, un error mínim en el disseny.
Tot i que el treball realitzat ha sigut només passar els tests al disseny, ha sigut molt eficaç ja que l'execució és ràpida i en una passada es va trobar l'error de les divisions.

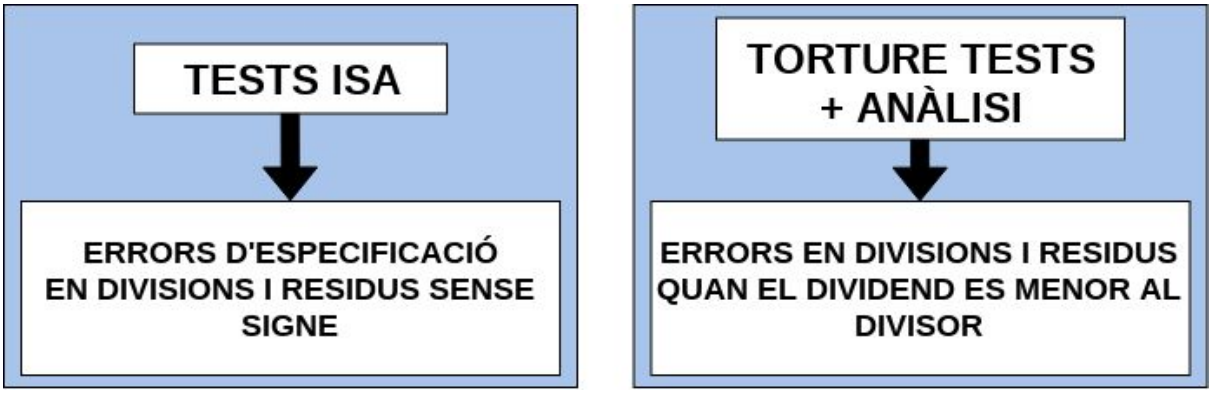

Figura 17: Diagrama representatiu dels errors trobats en la verificació

Més endavant, quan es van començar a utilitzar els torture test en grans quantitats, es va veure que un percentatge dels mateixos fallaven. Com ja s'ha explicat però, es va descobrir que els tests eren poc complexos i va ser una de les solucions, crear tests més curts, la que va fer que trobéssim un altre error en les divisions.
En aquest cas, els errors estaven relacionats amb certes combinacions de operands en les instruccions de divisions, una vegada més. Aquests errors van requerir més treball d'arreglar i una revisió a fons de l'especificació de l'ISA, però es van solucionar.

En el cas dels torture tests sí que hi va haver, a part de la simple execució dels mateixos, molt treball extra d'anàlisi tant dels tests com dels resultats. Es van analitzar els tests, en quant a estructura i resultat, i es van produir codis que eren capaços de solucionar

els problemes que tenien i analitzar els resultats. La combinació dels tests i el treball del verificador va fer possible la localització dels *bugs*.

No sabem exactament en quin punt del codi del kernel de Linux s'utilitzen o si tenen alguna cosa a veure realment, però després d'arreglar els errors trobats en aquest procés de verificació Linux fa *boot* sense problemes (abans es penjava en el *boot*, sense completar-lo) i permet utilitzar funcionalitats bàsiques del sistema operatiu.

Ja que també s'han fet altres modificacions en el RTL de la perifèria del processador en temes d'interfícies en memòria, no podem assegurar que els errors trobats siguin clau en aquesta millora del processador. El que és clar, però, és que qualsevol error del processador s'ha d'arreglar per donar el millor disseny possible.

7.2 Canvi a la estratègia de verificació

Durant el desenvolupament del projecte, l'estratègia de verificació ha sofrit unes variacions. Tot i que originalment el pla era fer primer tests bàsics, després passar a un tests més avançats per posteriorment trobar les raons del errors trobats utilitzant tests específics per tal d'aportar el feedback més complet possible als dissenyadors i un cop es solucionin els errors tornar a executar els tests avançats.

Aquest cicle es va repetint i permetria trobar errors i donar feedback concret als dissenyador mentre es solucionen els anteriors.
Això, però, no ha sigut així degut a que els dissenyadors tenien treball a realitzar relacionat amb altres parts del projecte.

Per tant, el desenvolupament de la verificació va ser diferent. La primera iteració del bucle va ser semblant; als tests ISA van aparéixer certs *bugs* que es van resoldre immediatament, però quan es va passar als torture tests i es van trobar errors els dissenyadors no estaven disponibles.

És per això que es va fer tant de treball amb els torture tests i es van tractar tant en profunditat, creant codis d'anàlisi i de facilitació del *debugging*, per tal de donar el feedback més concret possible.
Inicialment la idea era només trobar tests que fallaven i poder replicar les fallades però al final no només es trobaven els tests, també els errors que provocaven la fallada i per consegüent els operands i tot el que permet solucionar-los.

A més com que la revisió dels errors és cada més temps, té més sentit fer els tests ISA cada cop que hi ha un canvi per que són més ràpids i facilitan trobar errors bàsics que s'hagin pogut produir per les modificacions.

D'aquesta manera el cicle de la verificació ha canviat lleugerament. De l'original ja comentat, passa a ser:

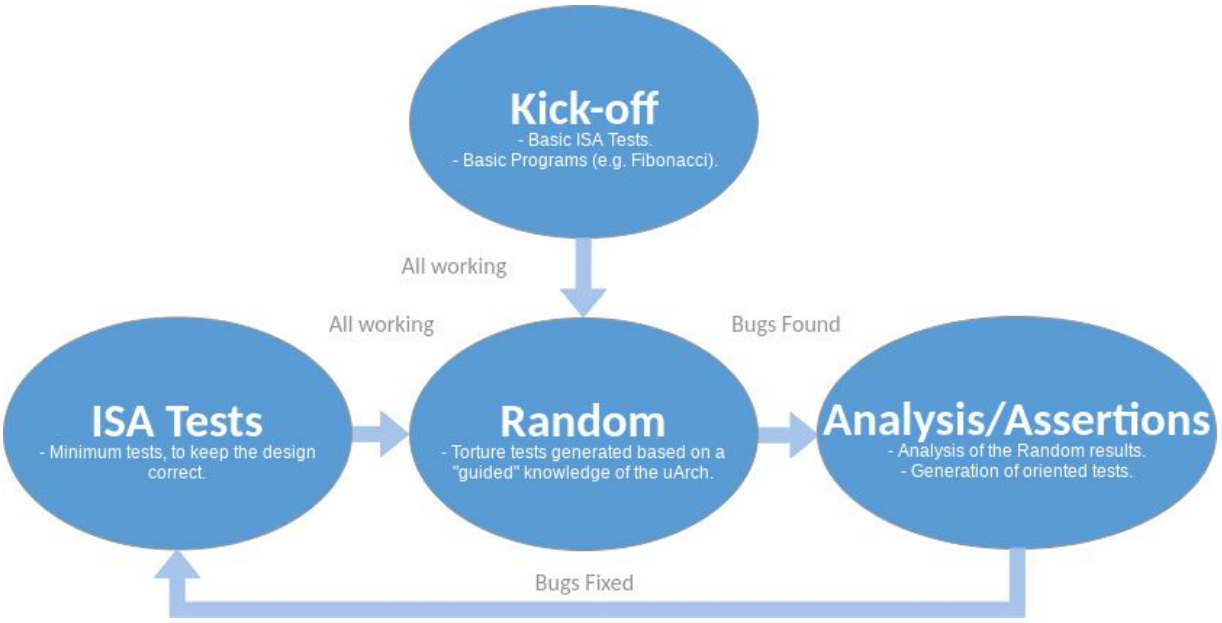

Figura 18: Diagrama de la nova estratègia de verificació

Aquesta metodologia ha sigut especialment eficient degut a que teníem les eines essencials, a que el processador estava pràcticament perfecte i a que els errors que ha presentat estaven bastant concentrats en el mateix mòdul. Gràcies a aquesta última circumstància hem pogut donar un feedback molt complet als dissenyadors realitzant l'anàlisi de molts tests i trobant-ne els errors.

La planificació inicial, això sí, s'ha vist totalment trastocada.

8 Treball futur

8.1 Execució pas a pas

Una de les eines essencials pel desenvolupament de la verificació ha sigut la de poder treure una porció de la memòria, anomenada *signature*, i veure el resultat dels tests mitjançant el bolcat dels registres en aquesta secció de memòria.

El gran problema d'aquesta funcionalitat és que només disposem de les dades després de l'última instrucció.
Per solucionar això ja s'han comentat dues solucions parcials, fer tests petits per poder veure tots els resultats i que no hi hagi sobreescriptures i fer diferents particions per poder fer més comprovacions.

Aquestes solucions aporten, però no solucionen tots els problemes.
Per exemple, si hi ha una escriptura errònia d'una instrucció X al registre x1, a la instrucció X + 1 s'escriu al registre x2 utilitzant com a operand el registre x1, donant un altre resultat erroni, i finalment aquest registre es sobreescriu a la instrucció X + 2 per donar un resultat correcte, tenim a la *signature* el registre x1 correcte i l'x2 incorrecte.

No només tenim, aparentment, un error en una instrucció (la que canvia el registre x2) que no és incorrecta, sinó que també ens perdem un error molt recent que provoca l'error realment inexistent anterior.

Aquest problema seria un gran impediment de cara al *debugging* ja que el que ens mostrarien els programes mencionats seria que hi ha un error en el registre x2 a la instrucció X + 1 (l'última modificació) i per tant els dissenyadors encarregats de trobar la causa de l'error perdrien molt de temps buscant on no és.

Per solucionar aquest inconvenient existeixen tècniques molt comuns anomenades *step-by-step* (pas a pas) que permeten veure els resultats al acabar cada instrucció (tota la *pipeline* del processador). Això arreglaria el cas explicat anteriorment degut a que veuriem el canvi erroni al registre x1 i podríem entendre al següent pas que la raó per la que el registre x2 té un error és perquè un dels operands ja és erroni.

El problema en el nostre cas és que Verilator i Spike, durant la verificació, no tenien aquestes opcions disponibles o almenys les nostres versions no disponien d'execució pas a pas.
És per això que es crea una solució temporal i molt ineficient cap al final del procés de verificació que permet emular aquesta funcionalitat.

El que es crea és un codi capaç de veure on comença i on acaba el codi real del test i fer-ne un test per instrucció. D'aquesta manera si un test comença a la línia 60 i acaba a la 460 hi ha 401 instruccions a testejar, pel que el programa escrit genera 401 tests.

Aquests tests estan tots composats pel que hi ha en les 60 primeres línies, el codi que pertoqui i el que hi ha després de la línia 460.
El codi que pertoca és, pel primer test (diguem-ne "test_1") la primera instrucció, pel segon test, "test_2", la primera i la segona instrucció i així successivament fins arribar al "test_401", que és exactament igual que el test inicial.

Això permet veure el resultat de cada instrucció, a més dels operands sense ser modificats.
El punt negatiu? La generació bruta de 401 arxius innecessaris i que l'execució triga al voltant de 401 vegades més que l'execució d'un simple test, degut a que tot i que els primers tests tenen 400 instruccions menys, l'*overhead* inicial i final segueix present i fa que les execucions siguin bastant lentes.

Tot i així, si disposem d'un test que falla i no sabem perquè, val la pena perdre el temps que faci falta si acabem trobant l'error.
També podem parar l'execució al 320 si no tenim temps d'acabar i tornar a començar a partir del 321 o si sabem que un test falla en les instruccions entre la 300 i la 309 només executar aquests 10 tests.

D'aquesta manera podem executar els tests i el primer que falli és on l'última instrucció és la problemàtica i només cal mirar aquesta.

Aquest procés queda fora del contingut present del projecte degut a que no es pot utilitzar eficientment i a que no hem trobat cap error utilitzant torture tests durant les últimes etapes del projecte, però podria arribar a ser útil i seria interessant desenvolupar una versió millor.

Per altra banda, durant les últimes setmanes abans de la primera entrega del processador, els dissenyadors van crear algunes eines per comunicar-se amb el processador durant l'execució i per fer execució pas a pas. Tant Spike com Verilator han anat afegint aquesta utilitat i s'ha pogut provar amb resultats exitosos.

Al disposar d'aquestes eines, en futurs projectes es podria fer un procés similar a l'actual però per cada instrucció.
L'execució pas a pas tant d'Spike com de Verilator permet agafar alguns detalls com el *program counter* i el codi de la instrucció, pel que la *signature* al acabar cada instrucció pot ser més completa i facilitar encara més el *debugging*.

L'objectiu seria utilitzar la comanda "diff" de Linux tal com fem ara per comparar les *signatures* d'ambdues execucions i això ens permetria veure tots els registres en totes les instruccions i enviar als companys encarregats d'arreglar un possible *bug* un missatge de l'estil "a la instrucció amb PC igual a 3C4, que és un addi x2, x3, 1234, hi ha una diferència

entre Verilator i Spike", que només haurien d'executar el test i anar directament a aquesta instrucció.

8.2 UVM

Com ja s'ha comentat en apartats introductoris del treball, l'estàndard en quant a verificació avui dia és UVM. Degut als requeriments del projecte i de la verificació en sí, UVM no s'adapta a les nostres necessitats, pel que fem una barreja de tècniques i "filosofies" per desenvolupar la nostra estratègia de verificació.

El cas però, és que UVM és una estratègia ben establerta i definida, utilitzada comunament en els projectes similars a Lagarto.
És per això que en un futur, idealment s'hauria d'adaptar el projecte a l'estratègia de verificació, ja que seria més senzill degut a la documentació i eines ja existents per utilitzar UVM.

Per això cal formació en aquesta metodologia i en algunes de les eines que aquesta utilitza, però s'ha d'anar tractant. Personalment, en el desenvolupament d'aquest treball he estudiat UVM per parlar-ne una mica en els primers apartats.

De totes maneres, no cal entendre del tot la metodologia per utilitzar certes funcionalitats de la mateixa.
És l'exemple d'uns tests basats en la metodologia UVM i que són semblants als torture tests en quant a que la seva generació és aleatoria. El codi de la seva generació i les seves diferents configuracions es troben en un repositori obert de Google a Github anomenat "riscv-dv"[23].

Al principi de la verificació es van trobar aquests tests però no es disposava de les eines necessàries per la seva generació, ja que requereixen eines UVM contingudes en software com *Cadence*.
Generant aquests tests es produeix una funcionalitat similar als torture tests però aquests tests UVM disposen de més complexitat en quant al testejat i en quant a l'estructura del programa. Tenen rutines de servei a interrupcions i excepcions a més de tests complexos de memòria virtual i més, pel que són uns bancs de proves més complexos.

Per tant, en un futur projecte aquests tests s'utilitzaran per davant dels torture tests per ser més complets directament.
Tot i així, es faran esforços per implantar UVM i explotar aquesta metodologia que amb temps i coneixement suficient de la mateixa podria donar molt bons resultats.

8.3 Pipeline Gitlab

Durant la utilització del repositori comú a Gitlab, es va descobrir l'eina *Pipeline*[24]. Aquesta eina permet l'automatització de certes comandes en relació amb el respositori ja que a cada "*push*" s'executa un *script* que pot utilitzar el que hi ha a les carpetes del repo. L'*script* s'ha d'anomenar ".gitlab-ci.yml"

L'exemple més bàsic és el que necessitem nosaltres; a cada *push* al repo d'un codi, s'executen les comandes que realitzen el testejat del nou codi.

En el nostre cas, a cada *push* del repo del codi del processador, s'executen les comandes responsables de l'execució de, per exemple, els torture tests.

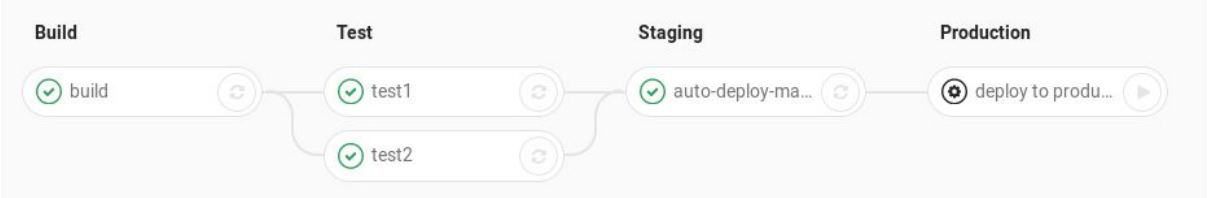

Figura 19: Exemple d'una pipeline i les seves etapes en la interfície gràfica[24]

Això és especialment útil al nostre projecte ja que hi ha bastant gent treballant en diferents parts del processador i això ens dóna la possibilitat de fer tests "ràpids" a cada canvi per veure si una millora en una part ha desfet alguna cosa que funcionava.

Pipeline significa canonada, que és la filosofia que segueix aquesta funcionalitat. Bàsicament es composa de diferents *stages* o etapes que s'executen unes després de les altres (o en paral·lel) i que tenen certes relacions de depèndencies entre elles.

En l'execució d'aquesta *pipeline* i utilitzant una interfície visual (exemple mostrat a sobre) que ens dóna Gitlab, podem veure el progrés de la nostra seqüència de comandes; si ha estat exitosa l'etapa X, si s'ha cancel·lat la *pipeline* o si no ha acabat encara.

A dins de cada etapa hi ha d'haver almenys un *job*, que és la secció que realment conté les comandes a executar. En l'exemple anterior de *pipeline*, l'etapa seria, per exemple, "Test" i els seus *jobs* "test1" i "test2".
La interfície anteriorment mencionada permet a més entrar a veure l'execució en terminal (comanda i sortida) de cada *job*.

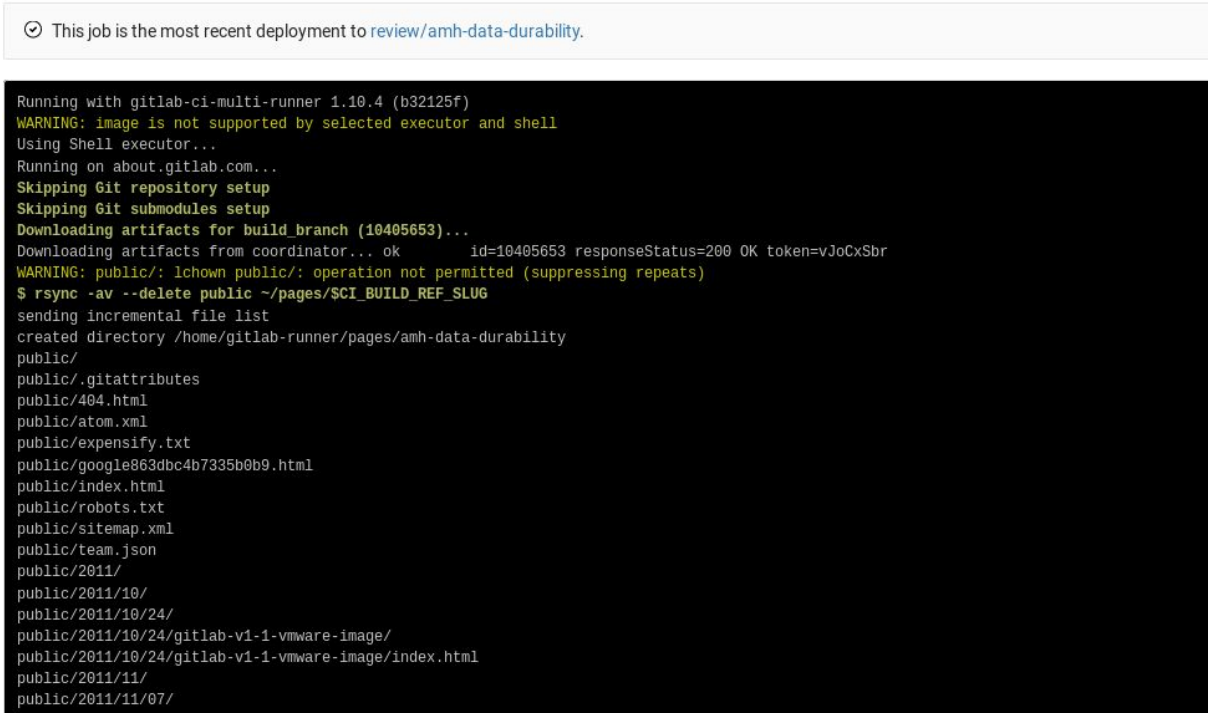

Figura 20: Exemple d'una execució (comanda i sortida) d'una etapa d'una *pipeline*[25]

Com ja s'ha mencionat, aquesta funcionalitat es basa en etapes i dependències. Si l'etapa B depèn de l'etapa A, aquesta última falla (sempre que hi ha un error en la comanda, qualsevol, es tracta com una falla), no s'executa l'etapa B. De fet, si qualsevol de les etapes falla, tota la *pipeline* falla.
Mitjançant codi però, es pot especificar que una etapa pot fallar i llavors només dóna un avís.

```
image: node:8.10
stages:
  - deploy
  - confidence-check
deploy_terraform:
  stage: deploy
  script:
    # Your Review App deployment scripts - for a working example please check https:/
e2e:firefox:
  stage: confidence-check
  services:
    - selenium/standalone-firefox
  script:
    - npm run confidence-check --host=selenium__standalone-firefox
e2e:chrome:
  stage: confidence-check
  services:
    - selenium/standalone-chrome
  script:
    - npm run confidence-check --host=selenium__standalone-chrome
```

Figura 21: Exemple de codi de l'arxiu .gitlab-ci.yml[26]. A la secció "*stages*" es declaren les diferents etapes i quan es declaren els *jobs*, com a "deploy_terraform", s'especifiquen les comandes a executar a "*script"* i s'inclou una variable "*stage*" en la que s'indica a quina etapa pertany. D'aquesta manera s'ordenen els *jobs* a la *pipeline*.

El codi a més permet guardar durant un cert període de temps certs arxius especificats anomenats "*artifacts*". Això és rellevant donat que una de les característiques més interessants d'aquesta funcionalitat és que tots els arxius (excepte els mencionats anteriorment) són borrats, pel que és una execució bastant neta.

A la interfície gràfica es pot veure etapa per etapa l'execució i es poden descarregar aquests *artifacts*[27]. Així, si executéssim els torture tests i guardéssim les *signatures* tindríem els resultats disponibles tal i com fem en les execucions locals.

De moment però, s'han fet proves només executant tests ISA i torture tests amb resultats interessants. Tot i que no hi ha tests erronis, s'ha conseguit forçar un error per veure el resultat de la *pipeline* a la interfície gràfica i s'han pogut extreure els *artifacts*.
La idea seria tenir una etapa de compilació tant del simulador com dels tests, una d'execució dels torture tests, per exemple, i una que utilitzés els codis mencionats en aquest treball per analitzar els resultats.

En un futur projecte serà molt útil des d'un principi poder provar el nucli del processador d'aquesta manera, potser recuperant els *artifacts* només quan els tests fallin per permetre el *debugging* des de el principi.
Aquesta funcionalitat també permetrà llançar, cada nit i en un servidor, l'execució de milers i milers de tests.

9 Conclusions finals

Durant el desenvolupament d'aquest projecte no només he realitzat la verificació funcional del processador donat, sinó que he adquirit molts coneixements i experiència realment importants per mi.
En quant als coneixements, a més de tècniques molt útils tant pel present com el futur i d'aprendre de professionals molt hàbils, també he vist i après com es dissenya i funciona un processador segmentat, que m'interessa molt personalment.

Amb el disseny donat i havent de fer la verificació, vaig descobrir eines que s'utilitzen sovint en projectes d'aquest estil per tal d'aquesta finalitat, com per exemple Spike i Verilog.
Cap al final de la verificació ja sabem utilitzar i treure el màxim partit a aquestes eines i les seves opcions, essent el procés d'implantació molt més ràpid si les volem utilitzar en un altre projecte, cosa realment probable ja que són eines molt interessants.
Una altra eina realment important és Docker, que té moltes possibilitats i s'utilitza en molts entorns, pel que haver après, en cert grau, a utilitzar-la ha sigut un punt molt positiu del projecte.

Per entendre, donar feedback complet i automatitzar processos s'han utilitzat programes creats específicament per aquest projecte.
Aquests codis han sigut escrits en Python i "bash script", scripts de Linux, i són la meva primera experiència en aquests tipus de programació, pel que he hagut d'aprendre i auto-formar-me en aquests camps, que em semblen realment útils. En el cas dels codis d'aquest projecte han sigut realment satisfactoris quan han funcionat i han sigut útils.

És clar que aquestes eines i aprenentatges m'han aportat una millora com a enginyer molt important, però el que ha aportat moltíssim ha sigut l'experiència de treballar en grup en un projecte i de tenir rellevància en el mateix.
És la primera vegada que treballo en un equip tan gran i a més amb gent tant treballadora i formada com el cas dels companys al BSC.

En relació amb aquesta experiència, em quedo amb la importància del procés de verificació en projectes d'aquest estil. Normalment, se'ns ensenya com funciona un processador i de vegades com es dissenya, pero no se'ns forma en com fer una bona verificació sinó només a que s'ha de provar el que s'ha fet.
En empreses com Intel, l'equip de verificació és igual de gran (i important) o més que el de disseny, perquè és realment important que el disseny funcioni correctament. Això demostra la importància del procés de verificació.

Per acabar, dir que estic orgullós tant del progrés i el procés de verificació com de la meva millora personal. A més puc dir que estic content amb els resultats, tot i que òbviament hi ha coses a millorar que en un futur, amb les coses que s'han après i amb l'experiència obtinguda, seran fàcilment resoltes.

10 Descripció de les tasques

En aquesta secció es detallen les tasques que comprenen el projecte. En la descripció d'aquestes també es tractaran breument els recursos i requeriments necessaris per a la realització de les mateixes. Els recursos s'especificaran més detalladament en un apartat més endavant.
A més, en el diagrama de Gantt[30] de l'apèndix A es troba la planificació final del projecte.

10.1 Gestió del projecte

Durant la fita inicial del projecte, al curs de Gestió de Projectes, s'han de realitzar un total de quatre lliuraments. El repartiment del temps de dedicació per a cada un dels mateixos és el següent:

- **Abast del projecte i contextualització:** 24,50h
- **Planificació temporal:** 8,25h
- **Gestió econòmica i sostenibilitat:** 9,25h
- **Presentació oral i document final:** 18,25h

Aquesta fase comprèn al voltant d'un mes, encara que es realitza paral·lelament amb d'altres.
Per a la realització d'aquesta fase del projecte es necessitaran un ordinador, Smartsheet, Google[31], Google Drive i les eines Google Docs i Presentacions de Google.

10.2 Estudi i aprenentatge de l'entorn de treball

Abans de començar amb el procés de verificació, es necessita conèixer a la perfecció tant el disseny del processador, escrit en Verilog, com l'entorn que s'utilitzarà per fer les proves. A més tot i que es tenen nocions bàsiques, fa falta practicar Verilog per entendre el disseny.

Per altra banda, per familiaritzar-se amb l'entorn de simulació, els companys tenen tests senzills que es poden executar. Això permet entendre com funcionen els tests i els resultats que donen.

El temps aproximat dedicat a aquesta fase no hauria de superar les tres setmanes.

Els recursos necessaris per aquesta fase del projecte serien un ordinador, el sistema operatiu Linux, Github, Gitlab i les eines de l'entorn de verificació, explicades a l'apartat de recursos més endavant, a més de l'ajut dels companys i director i co-director del projecte.

10.3 Disseny de la metodologia

Mitjançant la recerca dels companys que ja hi treballaven i unes reunions, es decideix la metodologia a realitzar. Per això, els diferents components del grup de verificació expliquen diferents possibilitats i eines que han trobat.
Donades aquestes possibilitats i el temps del que es disposa s'estableix una metodologia concreta. Que s'utilitzarà en les següents fases del projecte.

En un període d'una setmana s'ha de decidir la metodologia.
Els recursos necessaris per aquesta fase del projecte són les documentacions de les estratègies de verificació proposades, un ordinador i la dedicació per part dels companys per trobar un model que ens funcioni.

10.4 Aplicació de la metodologia

Una vegada conegut tant el disseny com el mètode de testeig, es pot començar amb el procés de verificació. La metodologia escollida té una fase inicial i una fase posterior cíclica, pel que les subfases de l'aplicació de la metodologia es poden anar repetint.

Els recursos necessaris per aquesta fase del projecte són un ordinador amb Linux, la bateria de tests bàsics, Vim, l'eina de generació de tests aleatoris i les eines de l'entorn de verificació explicades a l'apartat de recursos més endavant.

10.4.1 Tests simples

S'executen tests inicials per comprovar les funcionalitats bàsiques del processador. Per això, es buscaran tests existents que provin les instruccions del llenguatge màquina RISCV sense arribar a estressar el disseny. S'adaptaran aquests jocs de proves a la nostra forma de testejar i s'executaran tots, ja que no haurien de trigar massa.

Tot i que es podrien localitzar errors en aquesta sub-fase, no s'espera que això passi. En cas que es ocorreguessin errors durant l'execució d'aquests tests, com en fases següents, s'haurien de notificar a l'equip de disseny per tal d'arreglar-los. Una vegada solucionats els problemes s'hauria de tornar a executar els tests per tal d'assegurar el correcte funcionament del processador.
El temps aproximat no hauria d'arribar a les dues setmanes.

10.4.2 Tests aleatoris

En aquesta fase el verificador generarà una gran quantitat de tests aleatoris capaços de forçar al màxim el disseny. S'agafaran els tests incorrectes per analitzar-los, treure'n estadístiques i trobar quins segments o instruccions els fan fallar. Per això s'haurà de trobar una manera de trossejar-los i obtenir resultats parcials de les execucions.
Amb això es podran determinar les causes dels errors per posteriorment crear tests específics que permetin trobar la causa del error en la següent sub-fase.

El temps aproximat de dedicació per aquesta fase en la primera iteració serà superior, ja que s'hauran de crear codis que permetin extreure les estadístiques i trossejar els tests. Al voltant de tres setmanes hauria de ser suficient per tenir la infrastructura.
La següent vegada que s'arribi a aquesta sub-fase només caldria executar els tests i analitzar els resultats amb les eines ja creades, pel que seria més ràpida; una setmana com a màxim.

10.4.3 Tests dirigits

En aquesta fase s'utilitzen els resultats obtinguts dels tests aleatoris i el seu anàlisi per cercar l'origen dels errors. Per això, detectant patrons i instruccions problemàtiques, es crean i executen tests que repliquin els errors trobats.

L'objectiu d'aquesta sub-fase és donar feedback complet i concret als dissenyadors per que siguin capaços de trobar i arreglar ràpidament els errors en el disseny.
Si aquesta fase és exitosa i els dissenyadors poden arreglar els errors, s'ha de tornar a provar el disseny, tornant a la sub-fase de tests aleatoris.

El temps aproximat de dedicació en aquesta fase seria com a màxim una setmana, ja que no es pot perdre massa temps en trobar un error a no ser que sigui vital d'importància en el funcionament del processador.

10.5 Memoria final

Una vegada assolit un nivell de cobertura de verificació adequat, es trencarà el cicle i es procedirà a la redacció de la memòria i a la preparació de la defensa davant el tribunal amb els resultats obtinguts.

El temps aproximat necessari serà al voltant de dues setmanes.
Els recursos necessaris seran un ordinador i la documentació realitzada durant les anteriors tasques.

11 Dependències entre tasques

Tasca	Dependències
Gestió del projecte	-
Estudi i aprenentatge de l'entorn de treball	-
Disseny de la metodologia	Estudi i aprenentatge de l'entorn de treball
Aplicació de la metodologia	Disseny de la metodologia
Tests simples	-
Tests aleatoris (1)	Tests simples
Tests dirigits (1)	Tests aleatoris (1)
Tests aleatoris (n)	Tests dirigits (n)
Tests dirgits (n)	Tests aleatoris (n)
Memoria final	Aplicació de la metodologia

Taula 1: Dependències entre les tasques del projecte

12 Recursos

12.1 Recursos hardware

- **Ordinador portàtil propi:** Acer Swift 3(R1)[32] amb un Intel Core i5-7200U, 8 GB de memòria RAM i 250GB SSD d'emmagatzematge utilitzat inicialment per treballar tant al BSC com fora de casa.
- **Ordinador portàtil proporcionat pel BSC:** Dell Latitude 7480 14"(R2) Corei7-7600U 2,8 GHz - SSD 256 GB RAM 8 GB QWERTY[33]
- **Monitor proporcionat pel BSC:** Monitor Dell 24 SE2417HG(R3)[34]

12.2 Recursos software

- **Sistemes operatius:** Windows 10(R4)[35] per ofimàtica i Linux(R5)[36] per a la verificació.
- **Control de versions:** Github(R6)[37] i Gitlab(R7)[38] per guardar i controlar versions del codi.
- **Editors de text:** Vim(R8)[39], utilitzat per escriure codi en Python o Bash Script en Linux, i Notepad++(R9)[40] utilitzat inicialment a Windows.
- **Planificació:** Per a la planificació temporal de la fita inicial s'utilitza l'eina Smartsheet(R10)[41].
- **Eines utilitzades en el procés de verificació:** Docker(R11)[42]; que crea l'entorn de simulació, Verilator(R12); capaç d'emular el processador, Spike(R13); eina capaç de simular el funcionament del llenguatge màquina RISCV, els tests bàsics del llenguatge màquina(R14) i una eina capaç de generar tests aleatoris pel llenguatge màquina RISCV(R15).
- **Eines generals utilitzades:** Eines com Google(R16) per cercar informació, altres eines de Google com Drive(R17) i Docs(R18) i Mozilla Thunderbird(R19).

12.3 Recursos humans

- **Director i co-director:** Ajudaran i faran seguiment de les tasques desenvolupades per a l'autor.

- **Altres treballadors del BSC:** La interacció amb alguns d'ells serà necessària per alguna de les tasques a realitzar, com ara amb els dissenyadors del processador.

- **Autor:** Encarregat de realitzar el projecte.

12.4 Altres recursos materials

- **LLoc de treball(R20):** Una sala comuna a diversos treballadors del BSC a l'edifici Nexus 2.

- **Taula(R21):** La taula de la que es disposa al lloc de treball.

- **Cadira(R22):** La cadira de la que es disposa al lloc de treball.

- **Internet i electricitat(R23):** Connexió a internet i la xarxa elèctrica disponible al BSC.

13 Estimació temporal

En el diagrama de Gantt de l'apèndix A es detalla la planificació temporal. En la taula següent s'estimen les hores, suposant que es dediquen unes **sis hores diaries** els dies laborals, i s'especifiquen els recursos de cada tasca.

Tasca	Temps estimat (hores)	Recursos
Gestió del projecte	162	R10
Estudi i aprenentatge de l'entorn de treball	78	R11
Disseny de la metodologia	30	R11
Aplicació de la metodologia	540	R11, R12, R13, R14, R15
Memoria final	66	-
Totes les tasques (recursos comuns a totes les tasques)	876	R1, R2, R3, R4, R5, R6, R7, R8, R9, R16, R17, R18, R19, R20, R21, R22, R23

*__Algunes tasques són en paral·lel__, com es pot veure al diagrama de Gantt de l'apèndix A.

Taula 2: Estimació d'hores i recursos per tasques.

14 Valoració d'alternatives i pla d'acció

És bastant possible que avancem més ràpid del planificat, pel que el nombre d'iteracions al diagrama de Gantt creixeria. També és possible que siguem incapaços de trobar cert error i per tant augmenti el temps necessitat per completar una iteració.
Per solucionar aquest problema, dins l'equip de verificació haurem d'aprendre a identificar quan un error és massa complex o massa difícil de replicar per tal de no perdre massa temps. Igualment, s'establirà un període màxim per a l'anàlisi dels errors.

Es treballarà en paral·lel amb l'equip de disseny per tal que es solucionin els errors trobats. Això pot resultar en que l'equip de disseny no pugui entregar el codi de manera ràpida i ens alenteixi les iteracions a l'equip de verificació.
Aquest problema no té solució fàcil, ja que som grups diferents i l'equip de disseny té més tasques a realitzar. És per això que és tan important analitzar els resultats dels tests per tal de donar el feedback més concret possible. De totes maneres, com que el projecte és cíclic, les desviacions no afectaran a la finalització del mateix.

Donada la naturalesa del problema a afrontar, la probabilitat de que qualsevol dels problemes mencionats anteriorment passin és bastant elevada, al voltant del 60%, ja que no sabem si els errors seran fàcils de localitzar o arreglar.

15 Desviacions respecte a la planificació inicial

La planificació inicial es basava en una estratègia de verificació que seria cíclica i amb duracions fixades.
Durant les primeres setmanes del projecte, la planificació es va seguir perfectament; tant les etapes d'aprenentatge i disseny de l'estratègia de verificació com la primera etapa de l'estratègia en sí, basada en tests bàsics, van transcórrer sense problemes, inclós la solució dels errors trobats.

En passar a les següents etapes, centrades en els torture tests, va haver més problemes. Es van executar els tests amb certs resultats però no es va poder treballar amb ells de la manera que es volia.
Com ja s'ha mencionat a l'apartat anterior, alguns factors, que ja es van preveure inicialment a GEP, com la poca disponibilitat dels dissenyadors i la necessitat d'estudiar millor les eines, en concret els torture tests, han fet variar l'estratègia de verificació i per tant la planificació.

Les circumstàncies mencionades van allargar l'etapa "Assertions" de l'estratègia inicial, ja que al no entendre bé els torture tests es van haver d'estudiar i generar programes per millorar la seva eficàcia, a més de programes capaços d'analitzar patrons i estadístiques dels tests generats per tal de donar millor feedback als dissenyadors.

Per tant, en comptes de parlar d'al voltant d'una setmana de l'etapa "Assertions" es va passar a prop de dos mesos.
Això sí, un cop es van trobar bé els errors i es van poder solucionar, la planificació es va seguir de manera bastant fidel, essent un cicle pràcticament uniforme.

A més, com que l'objectiu és la verificació del disseny i s'ha fet dins del mateix temps previst, no s'han incomplert els plaços, però sí l'estructura de la planificació.
De la mateixa manera, com que no s'ha treballat més temps del previst sinó que s'han mogut les hores d'una tasca a una altra i no s'ha hagut de fer cap inversió imprevista, el pressupost no ha sofrit cap variació.

16 Autoavaluació sobre sostenibilitat

Quan es parla de sostenibilitat, no només es parla de la problemàtica ambiental, també es parla d'economia i societat. És per això que crec que, tot i que tinc bastant coneixement de l'impacte mediambiental de les activitats humanes i com combatir-lo, donat que no tinc molta formació sobre economia i societat, no crec que tingui tanta informació com per dir que ho sé tot. Tot i així, crec que en l'àmbit de la informàtica puc treballar en els tres aspectes còmodament, ja que sóc usuari i "creador".

Em considero capaç de trobar idees per solucionar els problemes de sostenibilitat dels projectes en els que participo, tot i que no sempre poden ser les més eficients. Això es pot deure a que no tinc la mateixa conscienciació en els tres aspectes; conec i tinc molt presents els costos i impactes ambientals dels productes informàtics, però no sempre penso en l'accessibilitat o ergonomia dels mateixos. A més, mai havia realitzat cap càlcul o estimació dels costs econòmics per estudiar-ne la viabilitat abans del projecte present.

Donada la poca experiència que tinc, crec que tinc moltíssim a aprendre en tots tres aspectes en els propers projectes que realitzi. Responent l'enquesta m'adono de la poca consciència que tinc respecte al bé que el meu treball pot fer a la societat si es té present quan es treballa. Termes com justícia, gestió econòmica o accessibilitat no són conceptes que, sent realista, hagi tingut mai massa en ment mentre treballava, i considero que potser hauria de parar-me a revisar la meva activitat més freqüentment.

17 Gestió econòmica

17.1 Identificació i estimació dels costos

Recurs Humà	Dedicació (hores)	Cost/hora (€/h)	Cost total (€)
Director de projecte	162	26,22	4.247,64
Verificador	714	20,25	14.458,5
Total	**876**	**46,47**	**18.706,16**

Taula 3: Costos dels recursos humans

El projecte es comprèn entre el 13 de gener i el 24 de juny, el que fa més o menys 23 setmanes a unes 6 hores al dia (dies laborables), al voltant de 714 hores. S'ha de tenir en compte que el curs GEP es fa en paral·lel amb altres tasques del projecte.

Per altra banda, a la cita mencionada, es troba el cost/hora de mercat d'un verificador hardware a Espanya, 15 €/hora bruts[43], pel que queda afegir costs extra com seguretat social i formació, pel que s'agreguen en un multiplicador d'1,35, resultant en 20,25 €/hora. Un càlcul similar s'utilitza amb el càrrec de director de projecte.

Recursos Software	Preu (€)	Unitats	Vida útil	Amortització (€/h)
openSUSE Tumbleweed	-	1	3 anys	Preinstal·lat
Windows 10	145[44]	1	3 anys	Preinstal·lat
Github, Gitlab	-	1	-	-
Notepad++, Vim	-	1	-	-
Smartsheet	-	1	-	-
Docker	-	1	-	-
Bateria de tests bàsics	-	1	-	-
Generador de tests aleatoris	-	1	-	-
Verilator	-	1	-	-
Spike	-	1	-	-
Eines de Google	-	1	-	-
Mozilla Thunderbird	-	1	-	-
Total	-	-	**0**	**0**

Taula 4: Costos dels recursos software

Recurs Hardware	Cost (€)	Vida útil	Amortització (€/h)
Acer Swift 3[45]	760*	3 anys	0,126
Dell Latitude 14” 7480	1.339[46]**	4 anys	0,167
Monitor Dell 24 SE2417HG	129[47]	4 anys	0,016
Total	**2.229**	**-**	**0,309**

*Preu en el moment de la compra (2017)
**Preu del Dell Latitude 7490, la versió actualment en venta.

Taula 5: Costos dels recursos hardware

L’amortització es calcula dividint el preu total entre el temps establert segons l’horari laboral (8 hores/dia laborable) durant la vida útil estimada.
Per tant, l’amortització del portàtil Acer Swift 3 per exemple seria preu total entre hores laborals en 3 anys: Amortització=760/(3 (anys)*8 (hores/dia)* 251 (dies laborals))=0,126 €/h

Una altra part de les despeses derivades del projecte són les relacionades amb les instal·lacions i els desplaçaments, tot i que no són recursos directes del projecte.

Despesa indirecta	Cost mensual (€/mes)	Duració (mes)	Cost total (€)
Electricitat	0,756	6	4,536
Espai de treball	280	6	1680
Mobiliari	1,757	6	10,542
Aigua	1,771	6	10,626
T-jove 1 zona[48]	35	6	210
Total	**319,284**	**6**	**1.915,704**

Taula 6: Estimació de les despeses indirectes del projecte

Els costos de mobiliari es composen d'una taula i una cadira (150+60=210 aproximadament) i la seva amortització mensual tal com s'ha detallat en apartats anteriors suposant una vida útil de 10 anys. Respecte a l'espai de treball, és el cost aproximat de lloguer del metre quadrat a la zona pels 2 metres quadrats dels que es disposen (14€/m2*20=280€)[49] i arrodonint a l'alça per tal d'incloure assegurança. La t-jove té un cost de 105€ per 3 mesos, pel que simplement es divideix entre 3.

Pel que fa a l'aigua s'utilitza el cost anual promig a Barcelona per una familia (composta de quatre persones)[50](502€) entre quatre i en les 6 hores que s'estan al lloc de treball. Igualment s'aproximen els costos d'electricitat (il·luminació, portàtil i monitor) al voltant de (100W/1000)*(0,06€/kWh[51])=0,006€/hora.

17.3 Contingències i imprevistos

Per calcular les contingències, donat que el projecte, al ser cíclic, té una planificació volàtil, s'hauria de mirar a l'alça. És per això que, si s'hagués de d'establir un percentatge per aquest aspecte, hauria d'estar proper al 15%.
Tot i això, com que les altres despeses són fixes (a falta d'algun imprevist major i inusual), situaria aquest percentatge al 14%.

Aquest percentatge, com s'ha dit, es situa al 14% perquè degut a la planificació cíclica és possible que els treballadors del projecte haguem de fer més hores, pel que l'imprevist més esperat seria un augment de la despesa en recursos humans. Això suposaria un augment d'hores de treball de l'autor, sobretot, a més d'un augment de les hores que els supervisors passaran revisant el treball.

Per altra banda, podria haver algun event, com ara l'avaria del portàtil en el que es treballa o la necessària contractació d'altres treballadors per a la verificació, que suposaria un inconvenient major pel pressupost, però en un principi no s'esperen.

17.4 Pressupost general

En aquesta secció es presenten les despeses per tasques i els costos totals sense i amb IVA. Només hi apareixen aquells recursos que suposen una despesa i que, per tant, tenen impacte al pressupost del projecte.

	Unitats	Amortització (€/h)	Temps (hores)	Total (€)
Gestió del projecte			162	**4.247,64**
Dell Latitude 14" 7480	1	0.167		0
Monitor Dell 24 SE2417HG	1	0.016		0
Acer Swift 3	1	0.126		0
Verificador	-	20,25		0
Director del projecte	-	26,22	162	4.247,64
Estudi i aprenentatge de l'entorn de treball			78	**1.603,60**
Dell Latitude 14" 7480	1	0.167		13,026
Monitor Dell 24 SE2417HG	1	0.016		1,248
Acer Swift 3	1	0.126		9,828
Verificador	-	20,25		1.579,50
Disseny de la metodologia			30	**0**
Dell Latitude 14" 7480	1	0.167		0
Monitor Dell 24 SE2417HG	1	0.016		0
Acer Swift 3	1	0.126		0
Verificador	-	20,25		0
Aplicació de la metodologia			540	**11.101,86**
Dell Latitude 14" 7480	1	0.167		90,18

Monitor Dell 24 SE2417HG	1	0.016		8,64
Acer Swift 3	1	0.126		68,04
Verificador	-	20,25		10.935
Memòria final			66	**1.356,90**
Dell Latitude 14” 7480	1	0.167		11,022
Monitor Dell 24 SE2417HG	1	0.016		1,056
Acer Swift 3	1	0.126		8,316
Verificador	-	20,25		1.336,50
Costos directes				**18.310**
Costos indirectes				**1.915,704**
Total sense IVA				15.978,31
Contingència (14%)				2.236,96
Total amb IVA (21 %)				**22.040,48**

Taula 7: Taula del pressupost

17.5 Control de gestió

Les úniques desviacions en el pressupost que es consideren plausibles són en quant a recursos humans, en el cas que s’hagi de treballar més hores de les previstes. La millor manera de prevenir-les és acotar bé el treball a realitzar (fet en setmanes anteriors) i organitzar-se bé entre els companys. Si tot i així hagués algun problema, com que la resta de les despeses són molt fixes, només disposem de les contingències per solucionar les desviacions.

En relació a aquest últim punt, si hagués un imprevist dels mencionats en apartats anteriors, més “catastròfics”, no tenim forma de solucionar-los econòmicament. La solució en aquests casos seria rebaixar les condicions de treball ja sigui compartint equip amb companys o canviant d'instal·lacions, segons el problema que sigui.

S’utilitzaran tres indicadors[52] per mesurar o trobar les desviacions en el pressupost. La mà d’obra serà el principal àmbit on hi pot haver desviacions, pel que necessitarem anar controlant-ne els costos. Els costos fixos, tot i que en principi no han de variar, els podrem controlar mitjançant l’indicador esmentat. Finalment, seria interessant, un

cop acabat el projecte, utilitzar el total del pressupost per identificar possibles problemes de planificació.

- Desviament de la mà d'obra en consum =(consum estimat – consum real)*cost estimat
- Desviament total en costos fixos = total cost fix - total cost fix real
- Desviament total del pressupost = total pressupost estimat - total pressupost real

A més del total, aquests controls es faran per a cada tasca especificada en la planificació.

18 Viabilitat econòmica

Una vegada feta l'anàlisi econòmica i el pressupost, s'ha de valorar la viabilitat econòmica del projecte.

Donat que l'objectiu no és fer un processador potent ni perfecte, és desenvolupar un processador eficient i que utilitzi el ISA RISC-V, la naturalesa del projecte Lagarto és més formativa i de desenvolupament que competitiva. Per tant, per la banda del BSC, la viabilitat econòmica del projecte de la verificació en concret en el curt plaç és molta, donat que es necessita fer la verificació sí o sí i es disposen de fons suficients. A més fets els càlculs es demostra que és viable.

Per altra banda, tant com per l'autor del projecte com pel BSC, a llarg termini és realment viable. No només es fa la verificació si no que és forma l'equip per futurs projectes. Per tant, amb la inversió feta per la verificació se'n treu un processador verificat i un equip de verificació eficaç. La inversió feta es recuperarà en quant s'hagin de fer nous projectes o si aquest és realment un èxit.

19 Reflexió sobre la sostenibilitat

En aquesta secció es responen les preguntes que es plantegen a la matriu de sostenibilitat, fent una reflexió sobre l'impacte del projecte en els diferents aspectes tractats referents a la sostenibilitat.

	Fita	PPP	Vida útil
Ambiental	**I**	**¿Has estimat l'impacte ambiental que tindrà la realització del projecte?** Sí, l'impacte ambiental és el relacionat amb el hardware utilitzat i l'energia necessària per realitzar-lo. **T'has plantejat minimitzar l'impacte, per exemple, reutilitzant recursos?** Gran part del hardware utilitzat ha sigut anteriorment utilitzat per companys.	**¿Com es resol actualment el problema que vols abordar (estat de l'art)?** Empreses grans utilitzen grans quantitats d'energia en la generació i execució de tests. **¿En què millorara ambientalment la teva solució a les existents?** L'objectiu del projecte és dissenyar una metodologia que estalviï temps, pel que es consumiran menys recursos.
Econòmic	**I**	**¿Has estimat el cost de la realització del projecte (recursos humans i materials)?** Una de les primeres fases del projecte requereix la realització d'un pressupost, pel que s'han estimat els costos tan humans com materials.	**¿Com es resol actualment el problema que vols abordar (estat de l'art)?** Empreses grans utilitzen molts recursos en la generació i execució de tests. **¿En què millorarà econòmicament la teva solució a les existents?** L'objectiu del projecte és dissenyar una metodologia que estalviï temps, pel que es consumiran menys recursos. RISCV.
Social	**I**	**¿Què creus que t'aportarà a nivell personal la realització d'aquest projecte?** Tant acadèmica com professionalment és una gran oportunitat i una experiència molt interessant, ja que em permet treballar amb experts i aprendre multitud d'habilitats molt útils per un enginyer.	**¿Com es resol actualment el problema que vols abordar (estat de l'art)?** Empreses grans busquen el major benefici i el secretisme. **¿En què millorarà socialment la teva solució a les existents?** L'ús del llenguatge màquina lliure RISCV fomenta la investigació.
			¿Existeix una necessitat real del projecte? Pel BSC és indispensable, ja que és necessari provar el disseny.

Taula 8: Matriu de sostenibilitat (Apèndix B) només amb les preguntes i respectives respostes de la fita inicial.

	Fita	PPP	Vida útil
Ambiental	**F**	**Has quantificat l'impacte ambiental de la realització del projecte?** L'energia que consumeixen els dispositius utilitzats, a més dels materials dels mateixos. **Quines mesures has pres per reduir l'impacte?** Reutilització de dispositius. **Has quantificat aquesta reducció?** Els materials dels mateixos.	**Quins recursos estimes que s'utilitzaran durant la vida útil del projecte?** L'energia que consumeixen els dispositius utilitzats durant la verificació. **Quin serà l'impacte ambiental d'aquests recursos?** Tot i que s'intenta utilitzar el mínim d'energia possible, la generació de la mateixa té un impacte ambiental important.
		Si fessis de nou el projecte, podries realitzar-lo amb menys recursos? Els recursos ja s'han optimitzat al màxim.	**El projecte permetrà reduir l'ús d'altres recursos?** Al ser bastant eficient, l'estratègia de verificació redueix l'ús d'energia respecte a altres estratègies. **Globalment, l'ús del projecte millorarà o empitjorarà la petjada ecològica?** Degut a la seva eficiència, la millorarà.
Econòmic	**F**	**Has quantificat el cost (recursos humans i materials) de la realització del projecte?** Sí, al principi del projecte es va estimar i no hi ha hagut cap desviació. **Quines decisions has pres per reduir el cost?** Accions com reutilitzar hardware han ajudat a reduir costos. **Has quantificat aquest estalvi?** Sí, el cost del hardware reutilitzat.	**Quin cost estimes que tindrà el projecte durant la seva vida útil?** El cost de l'energia utilitzada en el procés de verificació. **Es podria reduir aquest cost per fer-lo més viable?** Si s'aconseguís reduir l'ús d'energia tenint dispositius que consumeixin menys, es podria reduir el cost relacionat amb aquesta energia.
		S'ha ajustat el cost previst al cost final? Sí, tot i que els costos de personal són aproximats, no hi ha hagut cap desviació. **Has justificat les diferències (lliçons apreses)?** Només el comentari dels costos de personal.	**S'ha tingut en compte el cost dels ajustaments/actualitzacions/reparacions durant la vida útil del projecte?** Tot i que el treball realitzat no necessitarà reparacions, s'han pensat millores i el cost econòmic serà tant el de l'energia utilitzada com el cost del personal encarregat en aplicar-les.
Social	**F**	**La realització d'aquest projecte ha implicat reflexions significatives a nivell personal, professional o ètic de les persones que hi han intervingut?** Sí, en quant a coneixements i a treball	**Qui es beneficiarà de l'ús del projecte?** Tant el BSC, que verificarà el correcte ús del seu processador, com el desenvolupador del projecte per tots els coneixements adquirits.

		en equip m'he adonat de quant havia d'aprendre.	**Hi ha algun colectiu que pugui veure's perjudicat pel projecte?** En principi no hi ha cap colectiu perjudicat. **En quina mesura?** No hi ha cap colectiu perjudicat.
			En quina mesura soluciona el projecte el problema plantejat inicialment? El soluciona de moment, ja que al acabar el projecte, no hi ha errors relacionats amb el nucli (Lagarto).

Taula 9: Matriu de sostenibilitat (Apèndix B) només amb les preguntes i respectives respostes de la fita final.

	Riscos	
Ambiental	**F**	**Podrian produir-se escenaris que fessin augmentar la petjada ecològica del projecte?** Si es necessitessin més hores de treball o verificació, s'utilitzaria molta energia, amb l'impacte ambiental que això comporta. A més, si els dispositius utilitzats tinguessin que ser reemplaçats per averia o altres raons, els materials i la seva extracció dels nous dispositius augmentarien la petjada ecològica del projecte.
Econòmic	**F**	**Podrian produir-se escenaris que perjudiquessin la viabilitat del projecte?** Si es necessitessin més hores de treball o verificació, s'utilitzaria molta energia i més hores de personal, que costarien més diners dels prevists.
Social	**F**	**Podrian produir-se escenaris que fessin que el projecte fos perjudicial per algun segment de la població?** Per la població general és poc possible, però per a altres treballadors del projecte Lagarto, si hagués alguna deficiència en la verificació i trobés errors que no existeixen realment, faria treballar en excés als dissenyadors.
		Podria crear el projecte algun tipus de dependència que deixés als usuaris en posició de debilitat? Al ésser l'únic mètode de verificació complet disponible al projecte Lagarto hi ha molta dependència del mateix.

Taula 10: Matriu de sostenibilitat (Apèndix B) només amb les preguntes i respectives respostes a l'apartat riscos de la fita final.

19.1 Impacte ambiental

En la realització de projectes d'àmbit tecnològic és realment important tenir en compte l'impacte ambiental, ja que es consumeixen gran quantitats d'energia i els recursos utilitzats no sempre són extrets o creats de la forma més sostenible possible. Això s'accentua encara més en els projectes informàtics.

Respecte a l'estimació de l'impacte ambiental directe d'aquest projecte, queda determinat pel hardware utilitzat, els recursos necessaris per a la seva fabricació i l'energia utilitzada. Els materials de fabricació generen residus durant la seva extracció pel que important aprofitar-los el màxim possible. L'energia, per altra banda, sol ser extreta de matèries no renovables, tot i que aquesta tendència està canviant lentament.

Per minimitzar l'impacte es poden reutilitzar recursos, com és el cas d'aquest projecte, en el que com que alguns dels dispositius utilitzats són del Barcelona Supercomputing Center, són reutilitzats d'altres projectes i d'altres companys. Reutilitzant recursos contribuïm (remotament) a reduir l'impacte ambiental de l'extracció de materials.

Actualment, el problema es soluciona per part de les grans empreses utilitzant grans quantitats de generadors i executors de tests en els dissenys realitzats. Això però no és respectuós amb el medi ambient ja que s'utilitzen molts recursos i es consumeixen grans quantitats d'energia. La solució proposada millorarà en aquest aspecte l'estat de l'art ja que la metodologia buscarà l'eficiència, necessitant menys màquines treballant en la verificació.

19.2 Impacte econòmic

En quant a l'impacte econòmic del projecte, no té impacte per a l'economia global o nacional, però sí que requereix certa inversió per part de l'autor i el BSC i repercuteix en l'entitat, ja que si és exitós s'estalviarà futurs costos en verificació, contractació i formació.

En la redacció d'aquest document s'estima el cost tant en recursos humans com materials, ja que es realitza un pressupost. Per tant es té en compte l'impacte econòmic en el projecte i es coneixen les limitacions o excedències econòmiques del mateix.

Les grans empreses utilitzen un gran nombre de recursos, tant treballadors com màquines, en la verificació, ja que s'ho poden permetre. La solució resultant del projecte millorarà clarament en aquest aspecte ja que, de ser exitosa, la verificació es realitzarà amb un nombre molt reduït de treballadors i màquines.
Cal mencionar també que el llenguatge RISCV és obert, pel que si surt bé, possiblement en un futur es fabricaran molts més processadors amb llenguatges màquina gratuïts.

19.3 Impacte social

L'impacte social en projectes tecnològics, el benefici que la societat en treu, és molt gran quan els resultats van directes a consumidors o usuaris. En aquest cas els resultats poden afectar només en l'àmbit tecnològic, pel que l'impacte en la societat serà pràcticament nul.

En l'actualitat, les empreses productores de processadors busquen el secretisme i el benefici propi, utilitzant els seus propis llenguatges màquina. L'objectiu del projecte Lagarto és realitzar un processador utilitzant RISCV, un llenguatge màquina que tothom pot utilitzar. Si el projecte és un èxit, podria comportar un canvi de mentalitat en les empreses. Per tant la solució milloraria aquest aspecte de la tecnologia d'ocultar els avenços pel seu propi bé.

Degut a la necessitat que té el BSC d'assegurar que el seu disseny funciona correctament abans de la construcció física del processador, és indispensable el projecte. Respecte a la societat, com s'ha mencionat anteriorment, si el projecte és realment exitós haurà guanyat una nova forma de treballar aplicable en un futur.

Finalment, respecte a mi, la realització d'aquest projecte resulta, tant acadèmica com professionalment, una gran oportunitat i una experiència molt interessant, ja que em permet treballar amb experts i aprendre multitud d'habilitats molt útils per un enginyer. A més, és la meva primera experiència laboral derivada del grau i en un camp que em resulta realment interessant, pel que és una situació molt positiva en tots els sentits.

20 Bibliografia

[1] University of British Columbia. (1998). RTL Design. [online] Disponible a: http://www.ece.ubc.ca/~edc/379.jan99/lectures/lec8.pdf [Visitat el 21 Feb. 2019].

[2] Universidad de las Palmas de Gran Canaria (n.d.). Tutorial Verilog. [online] Disponible a: http://www.iuma.ulpgc.es/~nunez/clases-FdC/verilog/Verilog%20Tutorial%20v1.pdf [Visitat el 21 Feb. 2019].

[3] RISC-V Foundation. (2019). RISC-V Foundation | Instruction Set Architecture (ISA). [online] Disponible a: https://riscv.org/ [Visitat el 21 Feb. 2019].

[4] BSC-CNS. (2019). BSC-CNS. [online] Disponible a: https://www.bsc.es/ [Visitat el 21 Feb. 2019].

[5] Aynsley, J. (2010). *UVM Verification Primer*. [online] Doulos.com. Disponible a: https://www.doulos.com/knowhow/sysverilog/uvm/tutorial_0/ [Visitat el 21 Feb. 2019].

[6] Doulos.com. (2008). Getting Started with OVM. [online] Disponible a: https://www.doulos.com/knowhow/sysverilog/ovm/tutorial_0/ [Visitat el 25 Feb. 2019].

[7] En.wikipedia.org. (2019). Accellera. [online] Disponible a: https://en.wikipedia.org/wiki/Accellera [Visitat el 21 Feb. 2019].

[8] Accellera.org. (2015). [online] Disponible a: https://accellera.org/images/downloads/standards/uvm/uvm_users_guide_1.2.pdf [Visitat el 25 Feb. 2019].

[9] Mozhikunnath, R. (2016). What exactly is the difference between OVM, UVM, and VMM verification methodologies, and what do they signify? - Quora. [online] Quora.com. Disponible a: https://www.quora.com/What-exactly-is-the-difference-between-OVM-UVM-and-VMM-verification-methodologies-and-what-do-they-signify [Visitat el 25 Feb. 2019].

[10] Disciplinedagiledelivery.com. (2017). INTRODUCTION TO DISCIPLINED AGILE DELIVERY (DAD). [online] Disponible a: https://www.disciplinedagiledelivery.com/introduction-to-dad/ [Visitat el 25 Feb. 2019].

[11] Linchpin SEO. (n.d.). A Beginners Guide To Understanding The Agile Method | Linchpin SEO. [online] Disponible a: https://linchpinseo.com/the-agile-method/ [Visitat el 25 Feb. 2019].

[12] Quiroga Esparza, J. (2019) Estrategia de verificación para el proyecto Lagarto.

[13] Veripool.org. (2019). Intro - Verilator - Veripool. [online] Disponible a: https://www.veripool.org/wiki/verilator [Visitat el 11 Juny 2019].

[14] GitHub. (2019). riscv/riscv-isa-sim. [online] Disponible a: https://github.com/riscv/riscv-isa-sim [Visitat el 11 Juny 2019].

[15] GitHub. (2019). riscv/riscv-tests. [online] Disponible a: https://github.com/riscv/riscv-tests [Visitat el 11 Juny 2019].

[16] GitHub. (2019). ucb-bar/riscv-torture. [online] Disponible a: https://github.com/ucb-bar/riscv-torture [Visitat el 11 Juny 2019].

[17] Ramírez Lazo, Cristóbal (2018) Diagrama del pipeline de Lagarto, Lagarto I.

[18] Anon, (2019). [online] Disponible a: https://www.researchgate.net/figure/A-2-bit-bimodal-predictor-Values-inside-each-state-represent-the-value-of_fig1_242399534 [Visitat el 13 Juny 2019].

[19] Lowrisc.org. (2019). Overview of the Rocket chip · lowRISC. [online] Disponible a: https://www.lowrisc.org/docs/untether-v0.2/overview/ [Visitat el 13 Juny 2019].

[20] Docker.com. (2019). [online] Disponible a: https://www.docker.com/sites/default/files/social/docker_facebook_share.png [Visitat el 14 Juny 2019].

[21] Docker.com. (2019). [online] Disponible a: https://www.docker.com/sites/default/files/d8/styles/large/public/2018-11/container-what-is-contain er.png?itok=vle7kjDj [Visitat el 14 Juny 2019].

[22] RISC-V Foundation. (2019). Specifications - RISC-V Foundation. [online] Disponible a: https://riscv.org/specifications/ [Visitat el 18 Juny 2019].

[23] GitHub. (2019). google/riscv-dv. [online] Disponible a: https://github.com/google/riscv-dv [Visitat el 21 Juny 2019].

[24] Docs.gitlab.com. (2019). Creating and using CI/CD pipelines | GitLab. [online] Disponible a: https://docs.gitlab.com/ee/ci/pipelines.html [Visitat el 21 Juny 2019].

[25] Docs.gitlab.com. (2019). Getting started with GitLab CI/CD | GitLab. [online] Disponible a: https://docs.gitlab.com/ee/ci/quick_start/ [Visitat el 21 Juny 2019].

[26] Docs.gitlab.com. (2019). End-to-end testing with GitLab CI/CD and WebdriverIO | GitLab. [online] Disponible a: https://docs.gitlab.com/ee/ci/examples/end_to_end_testing_webdriverio/index.html [Visitat el 21 Juny 2019].

[27] Docs.gitlab.com. (2019). Introduction to job artifacts | GitLab. [online] Disponible a: https://docs.gitlab.com/ee/user/project/pipelines/job_artifacts.html [Visitat el 21 Juny 2019].

[28] GitHub. (2019). Build software better, together. [online] Disponible a: https://github.com/ [Visitat el 25 Feb. 2019].

[29] GitLab. (2019). The first single application for the entire DevOps lifecycle - GitLab. [online] Disponible a: https://about.gitlab.com/ [Visitat el 25 Feb. 2019].

[30] En.wikipedia.org. (2019). Gantt chart. [online] Disponible a: https://en.wikipedia.org/wiki/Gantt_chart [Visitat el 27 Feb. 2019].

[31] Google.es. (2019). Google. [online] Disponible a: https://www.google.es/ [Visitat el 27 Feb. 2019].

[32] Acer. (2019). Swift 3 SF314-52. [online] Disponible a: https://www.acer.com/ac/es/ES/content/model/NX.GNUEB.005 [Visitat el 27 Feb. 2019].

[33] Dell.com. (2019). Latitude 14" 7480 Portátil para empresas | Dell España. [online] Disponible a: https://www.dell.com/es-es/work/shop/port%C3%A1tiles-dell/port%C3%A1til-latitude-7480/spd/latitude-14-7480-laptop [Visitat el 4 Mar. 2019].

[34] Dell.com. (2019). Monitor Dell 24 SE2417HG | Dell España. [online] Disponible a: https://www.dell.com/es-es/shop/monitor-dell-24-se2417hg/apd/210-aldy/monitores-y-accesorios [Visitat el 4 Mar. 2019].

[35] Microsoft.com. (2019). Explora las nuevas actualizaciones y características de Windows 10 | Ve cómo es Windows 10. [online] Disponible a: https://www.microsoft.com/es-es/windows/features [Visitat el 27 Feb. 2019].

[36] Ubuntu.com. (2019). The leading operating system for PCs, IoT devices, servers and the cloud | Ubuntu. [online] Disponible a: https://www.ubuntu.com/ [Visitat el 27 Feb. 2019].

[37] GitHub. (2019). Build software better, together. [online] Disponible a: https://github.com/ [Visitat el 27 Feb. 2019].

[38] Gitlab. (2019). Gitlab. [online] Disponible a: https://repo.hca.bsc.es/gitlab/ [Visitat el 27 Feb. 2019].

[39] Vim.org. (2019). welcome home : vim online. [online] Disponible a: https://www.vim.org/ [Visitat el 27 Feb. 2019].

[40] Notepad-plus-plus.org. (2019). Notepad++ Home. [online] Disponible a: https://notepad-plus-plus.org/ [Visitat el 27 Feb. 2019].

[41] Smartsheet. (2019). Smartsheet: Less Talk, More Action. [online] Disponible a: https://es.smartsheet.com/ [Visitat el 27 Feb. 2019].

[42] Docker. (2019). Enterprise Application Container Platform | Docker. [online] Disponible a: https://www.docker.com/ [Visitat el 27 Feb. 2019].

[43] Guiasalarial.hays.es. (2019). [online] Disponible a: https://guiasalarial.hays.es/descargar/sectores-y-salarios [Visitat el 5 Mar. 2019].

[44] Microsoft Store. (2019). Comprar Windows 10 Home - Microsoft Store es-ES. [online] Disponible a: https://www.microsoft.com/es-es/p/windows-10-home/d76qx4bznwk4/1NT3 [Visitat el 5 Mar. 2019].

[45] Acer. (2019). Swift 3. [online] Disponible a: https://www.acer.com/ac/es/ES/content/series/swift3 [Visitat el 5 Mar. 2019].

[46] Dell.com. (2019). Portátil Latitude 7490 de clase empresarial de 14 pulgadas | Dell España. [online] Disponible a: https://www.dell.com/es-es/work/shop/laptops/3556-cm-7490/spd/latitude-14-7490-laptop?stacks=true [Visitat el 5 Mar. 2019].

[47] Amazon, (2019). [online] Disponible a: https://www.amazon.es/DELL-SE2417HG-Monitor-Color-Negro/dp/B06Y1R7NKR [Visitat el 5 Mar. 2019].

[48] Tmb.cat. (2019). Precio tarjeta T-Jove metro bus Barcelona | Transports Metropolitans de Barcelona. [online] Disponible a: https://www.tmb.cat/es/tarifas-metro-bus-barcelona/sencillos-e-integrados/t-jove [Visitat el 5 Mar. 2019].

[49] idealista/news. (2019). Precio del metro cuadrado en Barcelona. [online] Disponible a: https://www.idealista.com/news/estadisticas/precio-linea-metro/barcelona [Visitat el 18 Mar. 2019].

[50] www.ocu.org. (2019). Precio del agua. [online] Disponible a: https://www.ocu.org/alimentacion/agua/informe/el-precio-del-agua [Visitat el 18 Mar. 2019].

[51] https://tarifasgasluz.com (2019). Precio kWh 2019: ¿Qué compañía ofrece la luz más barata?. [online] tarifasgasluz.com. Disponible a: https://tarifasgasluz.com/faq/precio-kwh [Visitat el 19 Mar. 2019].

[52] Ferran Sabaté, Manel Rajadell, Fernando Barrabés (2019) Mòdul 2.4 - Gestión económica.pdf, “Indicadores de desviación”

[53] Curs de gestió de projectes, FIB, UPC (2018) Mòdul 2.6 - El informe de sostenibilidad 2018.pdf, “Preguntas de la matriz de Sostenibilidad del TFG”

Apèndixs

A Taula i Diagrama de Gantt de la planificació

Nom de la tasca	Duració	Inici	Finalització	Predecesores	Recursos
Gestió del projecte	27d	18/02/19	26/03/19		Ordinador, Smartsheet, Google, Google Drive i les eines Google Docs i Presentacions de Google.
Abast del projecte i contextualització	7d	18/02/19	26/02/19		
Planificació temporal	4d	27/02/19	04/03/19	2	
Gestió econòmica i sostenibilitat	5d	05/03/19	11/03/19	3	
Document final	7d	12/03/19	20/03/19	4	
Presentació oral	4d	21/03/19	26/03/19	5	
Fita: Document Final GEP	1d	21/03/19	21/03/19	5, 6	
Estudi i aprenentatge de l'entorn de treball	13d	16/01/19	01/02/19		Ordinador, Linux, Github, Gitlab, Vim i les Eines de l'entorn de verificació
Disseny de la metodologia	5d	28/01/19	01/02/19		Documentacions de les estratègies de verificació
Aplicació de la metodologia	90d	04/02/19	07/06/19	8, 9	Ordinador, Linux, Bateria de tests bàsics, Vim, Eina de generació de tests aleatoris i les Eines de l'entorn de verificació
Tests simples	10d	04/02/19	15/02/19		
Tests aleatoris (0)	15d	18/02/19	08/03/19	11	
Tests dirigits (0)	5d	11/03/19	15/03/19	12	
Tests aleatoris (1)	5d	18/03/19	22/03/19	13	
Tests dirigits (1)	5d	25/03/19	29/03/19	14	
Tests aleatoris (2)	5d	01/04/19	05/04/19	15	
Tests dirigits (2)	5d	08/04/19	12/04/19	16	
Tests aleatoris (3)	5d	15/04/19	19/04/19	17	
Tests dirigits (3)	5d	22/04/19	26/04/19	18	
Tests aleatoris (4)	5d	29/04/19	03/05/19	19	
Tests dirigits (4)	5d	06/05/19	10/05/19	20	
Tests aleatoris (5)	5d	13/05/19	17/05/19	21	
Tests dirigits (5)	5d	20/05/19	24/05/19	22	
Tests aleatoris (6)	5d	27/05/19	31/05/19	23	
Tests dirigits (6)	5d	03/06/19	07/06/19	24	
Memòria final	11d	10/06/19	24/06/19	1, 10	Ordinador, Smartsheet, Google, Google Drive i les eines Google Docs i Presentacions de Google.
Fita: Entrega Memòria Final	1d	25/06/19	25/06/19	26	
Fita: Defensa Oral TFG	5d	25/06/19	01/07/19	26	

Figura 22: Taula del diagrama de Gantt que conté la informació sobre la planificació i les tasques del projecte.

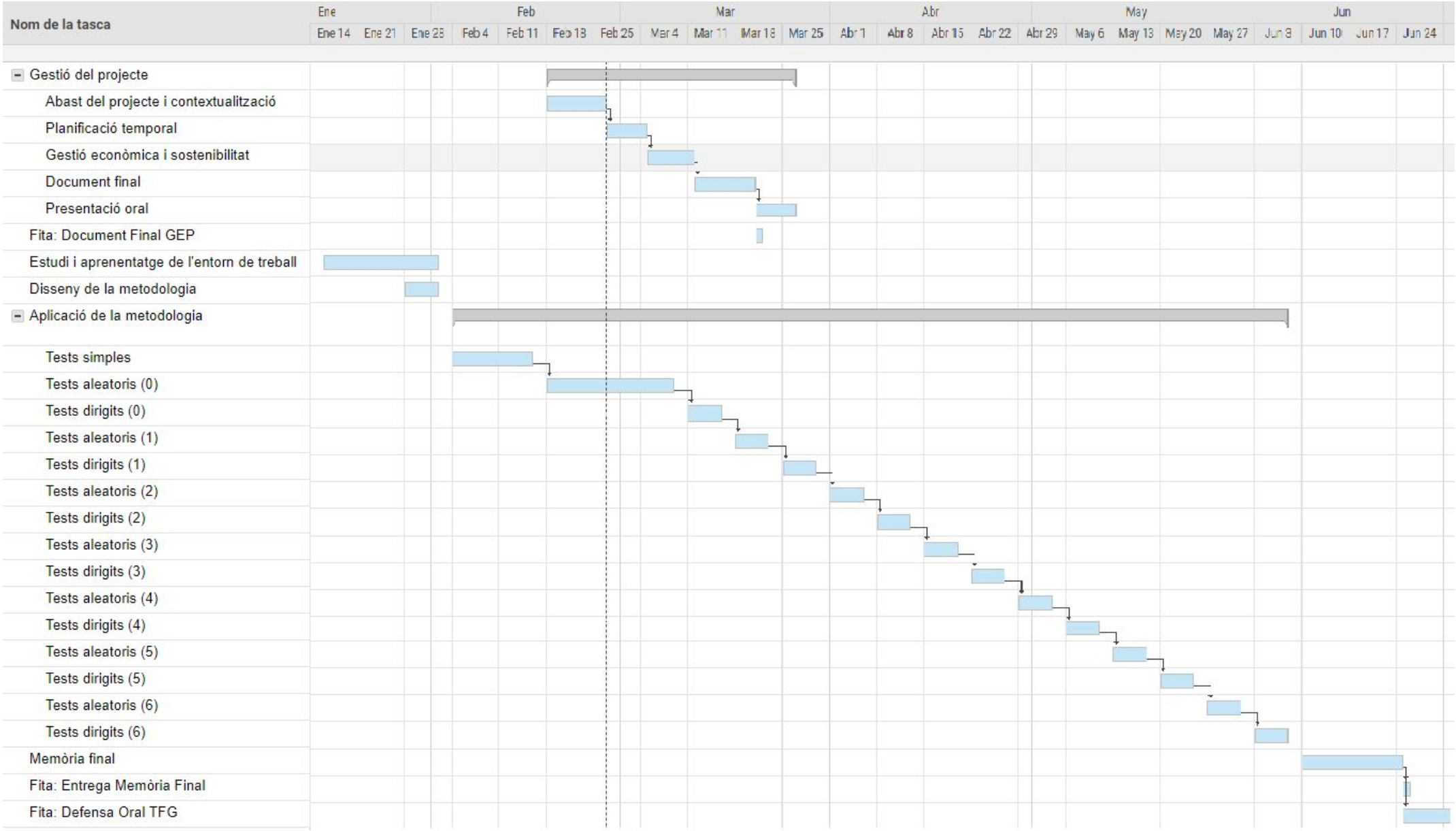

Figura 23: Diagrama de Gantt amb la planificació del projecte

B Matriu de sostenibilitat

		PPP	Vida Útil	Riesgos
Ambiental	I	¿Has estimado el impacto ambiental que tendrá la realización del proyecto? ¿Te has planteado minimizar el impacto, por ejemplo, reutilizando recursos?	¿Cómo se resuelve actualmente el problema que quieres abordar (estado del arte)? ¿En qué mejorará ambientalmente tu solución a las existentes?	
	F	¿Has cuantificado el impacto ambiental de la realización del proyecto? ¿Qué medidas has tomado para reducir el impacto? ¿Has cuantificado esta reducción?	¿Qué recursos estimas que se usarán durante la vida útil del proyecto? ¿Cuál será el impacto ambiental de estos recursos?	¿Podrían producirse escenarios que hiciesen aumentar la huella ecológica del proyecto?
		Si hicieras de nuevo el proyecto, ¿podrías realizarlo con menos recursos?	¿El proyecto permitirá reducir el uso de otros recursos? ¿Globalmente, el uso del proyecto mejorará o empeorará la huella ecológica?	
Económico	I	¿Has estimado el coste de la realización del proyecto (recursos humanos y materiales)?	¿Cómo se resuelve actualmente el problema que quieres abordar (estado del arte)? ¿En qué mejorará económicamente tu solución a las existentes?	
	F	¿Has cuantificado el coste (recursos humanos y materiales) de la realización del proyecto? ¿Qué decisiones has tomado para reducir el coste? ¿Has cuantificado este ahorro?	¿Qué coste estimas que tendrá el proyecto durante su vida útil? ¿Se podría reducir este coste para hacerlo más viable?	¿Podrían producirse escenarios que perjudicasen la viabilidad del proyecto?
		¿Se ha ajustado el coste previsto al coste final? ¿Has justificado las diferencias (lecciones aprendidas)?	¿Se ha tenido en cuenta el coste de los ajustes/actualizaciones/reparaciones durante la vida útil del proyecto?	
Social	I	¿Qué crees que te va a aportar a nivel personal la realización de este proyecto?	¿Cómo se resuelve actualmente el problema que quieres abordar (estado del arte)? ¿En qué mejorará socialmente (calidad de vida) tu solución a las existentes?	
			¿Existe una necesidad real del proyecto?	
	F	¿La realización de este proyecto ha implicado reflexiones significativas a nivel personal, profesional o ético de las personas que han intervenido?	¿Quién se beneficiará del uso del proyecto? ¿Hay algún colectivo que puede verse perjudicado por el proyecto? ¿En qué medida?	¿Podrían producirse escenarios que hiciesen que el proyecto fuese perjudicial para algún segmento particular de la población?
			¿En qué medida soluciona el proyecto el problema planteado inicialmente?	¿Podría crear el proyecto algún tipo de dependencia que dejase a los usuarios en posición de debilidad?

Figura 24: Preguntes completes de la matriu de sostenibilitat[53]

	Fita	PPP	Vida útil
Ambiental	**I**	**¿Has estimat l'impacte ambiental que tindrà la realització del projecte?** Sí, l'impacte ambiental és el relacionat amb el hardware utilitzat i l'energia necessària per realitzar-lo. **T'has plantejat minimitzar l'impacte, per exemple, reutilitzant recursos?** Gran part del hardware utilitzat ha sigut anteriorment utilitzat per companys.	**¿Com es resol actualment el problema que vols abordar (estat de l'art)?** Empreses grans utilitzen grans quantitats d'energia en la generació i execució de tests. **¿En què millorara ambientalment la teva solució a les existents?** L'objectiu del projecte és dissenyar una metodologia que estalviï temps, pel que es consumiran menys recursos.
Econòmic	**I**	**¿Has estimat el cost de la realització del projecte (recursos humans i materials)?** Una de les primeres fases del projecte requereix la realització d'un pressupost, pel que s'han estimat els costos tan humans com materials.	**¿Com es resol actualment el problema que vols abordar (estat de l'art)?** Empreses grans utilitzen molts recursos en la generació i execució de tests. **¿En què millorarà econòmicament la teva solució a les existents?** L'objectiu del projecte és dissenyar una metodologia que estalviï temps, pel que es consumiran menys recursos. RISCV.
Social	**I**	**¿Què creus que t'aportarà a nivell personal la realització d'aquest projecte?** Tant acadèmica com professionalment és una gran oportunitat i una experiència molt interessant, ja que em permet treballar amb experts i aprendre multitud d'habilitats molt útils per un enginyer.	**¿Com es resol actualment el problema que vols abordar (estat de l'art)?** Empreses grans busquen el major benefici i el secretisme. **¿En què millorarà socialment la teva solució a les existents?** L'ús del llenguatge màquina lliure RISCV fomenta la investigació. **¿Existeix una necessitat real del projecte?** Pel BSC és indispensable, ja que és necessari provar el disseny.

	Fita	PPP	Vida útil	Riscos
Ambiental	**F**	**Has quantificat l'impacte ambiental de la realització del projecte?** L'energia que consumeixen els dispositius utilitzats, a més dels materials dels mateixos. **Quines mesures has pres per reduir l'impacte?** Reutilització de dispositius. **Has quantificat aquesta reducció?** Els materials dels mateixos.	**Quins recursos estimes que s'utilitzaran durant la vida útil del projecte?** L'energia que consumeixen els dispositius utilitzats durant la verificació. **Quin serà l'impacte ambiental d'aquests recursos?** Tot i que s'intenta utilitzar el mínim d'energia possible, la generació de la mateixa té un impacte ambiental important.	**Podrien produir-se escenaris que fessin augmentar la petjada ecològica del projecte?** Si es necessitessin més hores de treball o verificació, s'utilitzaria molta energia, amb l'impacte ambiental que això comporta. A més, si els dispositius utilitzats tinguessin que ser reemplaçats per averia o altres raons, els materials i la seva extracció dels nous dispositius augmentarien la petjada ecològica del projecte.
		Si fessis de nou el projecte, podries realitzar-lo amb menys recursos? Els recursos ja s'han optimitzat al màxim.	**El projecte permetrà reduir l'ús d'altres recursos?** Al ser bastant eficient, l'estratègia de verificació redueix l'ús d'energia respecte a altres estratègies. **Globalment, l'ús del projecte millorarà o empitjorarà la petjada ecològica?** Degut a la seva eficiència, la millorarà.	
Econòmic	**F**	**Has quantificat el cost (recursos humans i materials) de la realització del projecte?** Sí, al principi del projecte es va estimar i no hi ha hagut cap desviació. **Quines decisions has pres per reduir el cost?** Accions com reutilitzar hardware han ajudat a reduir costos. **Has quantificat aquest estalvi?** Sí, el cost del hardware reutilitzat.	**Quin cost estimes que tindrà el projecte durant la seva vida útil?** El cost de l'energia utilitzada en el procés de verificació. **Es podria reduir aquest cost per fer-lo més viable?** Si s'aconseguís reduir l'ús d'energia tenint dispositius que consumeixin menys, es podria reduir el cost relacionat amb aquesta energia.	**Podrien produir-se escenaris que perjudiquessin la viabilitat del projecte?** Si es necessitessin més hores de treball o verificació, s'utilitzaria molta energia i més hores de personal, que costarien més diners dels prevists.

		S'ha ajustat el cost previst al cost final? Sí, tot i que els costos de personal són aproximats, no hi ha hagut cap desviació. **Has justificat les diferències (lliçons apreses)?** Només el comentari dels costos de personal.	**S'ha tingut en compte el cost dels ajustaments/actualitzacions/reparacions durant la vida útil del projecte?** Tot i que el treball realitzat no necessitarà reparacions, s'han pensat millores i el cost econòmic serà tant el de l'energia utilitzada com el cost del personal encarregat en aplicar-les.	
Social	**F**	**La realització d'aquest projecte ha implicat reflexions significatives a nivell personal, professional o ètic de les persones que hi han intervingut?** Sí, en quant a coneixements i a treball en equip m'he adonat de quant havia d'aprendre.	**Hi ha algun colectiu que pugui veure's perjudicat pel projecte?** En principi no hi ha cap colectiu perjudicat. **En quina mesura?** No hi ha cap colectiu perjudicat.	**Podrien produir-se escenaris que fessin que el projecte fos perjudicial per algun segment de la població?** Per la població general és poc possible, però per a altres treballadors del projecte Lagarto, si hagués alguna deficiència en la verificació i trobés errors que no existeixen realment, faria treballar en excés als dissenyadors.
			En quina mesura soluciona el projecte el problema plantejat inicialment? El soluciona de moment, ja que al acabar el projecte, no hi ha errors relacionats amb el nucli (Lagarto).	**Podria crear el projecte algun tipus de dependència que deixés als usuaris en posició de debilitat?** Al èsser l'únic mètode de verificació complet disponible al projecte Lagarto hi ha molta dependència del mateix.